PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERIA

TEMA:

DESARROLLO DE UN MODELO DE CLASIFICACIÓN PARA LA DETECCIÓN DE FRAUDES EN TRANSACCIONES DE TARJETAS DE CRÉDITO MEDIANTE EL USO DE TÉCNICAS DE DEEP LEARNING

AUTOR:

CARLOS ANDRES LINCANGO TITE

TRABAJO PREVIO A LA OBTENCIÓN DEL TÍTULO DE MAGISTER EN SISTEMAS DE INFORMACIÓN CON MENCIÓN EN DATA SCIENE

QUITO, AGOSTO – 2024

Tabla de contenido

Índice de Figuras.

CAPITULO I: INTRODUCCIÓN.

1.1 Introducción.

El presente trabajo de titulación es el fruto del conocimiento adquirido durante la Maestría de Sistemas de Información con mención en Data Science aplicados a una problemática específica: la utilización de técnicas de aprendizaje profundo para la detección de anomalías en transacciones generadas con tarjetas de crédito.

1.2 Justificación

De acuerdo con información de la Superintendencia de Bancos y Aval Buró, en 2023 en Ecuador existieron 4,2 millones de tarjetas de crédito activas, con las cuales, más de 2 millones de tarjetahabientes realizaron 105 millones de consumos por un monto de USD 21.891 millones. De esta manera, el número de consumos y el monto transaccionado a través de tarjetas de crédito creció frente a 2022 en 11,7% y 17,4%, respectivamente (Asobanca, 2024)

De Enero a Junio del 2023, 50.540 nuevos ecuatorianos accedieron por primera vez al sistema formal a través de una tarjeta de crédito. Ellos realizaron 192.999 operaciones. (Asobanca, 2023)

El mercado global de pagos con tarjeta de crédito alcanzara para el año 2025 los $ 2 billones, pero también se espera una pérdida como consecuencia del fraude de hasta $ 200 millones. (Gestion, 2021)

El tema expuesto en esta investigación es de vital importancia pues intenta desarrollar un modelo de clasificación a partir de aplicar técnicas de aprendizaje profundo que ayuden en la clasificación y la detección de fraudes y anomalías en transacciones ejecutadas con tarjetas de crédito.

La pertinencia de desarrollar esta investigación aplicada es la entregar resultados a partir de la utilización de técnicas avanzadas y que dichos resultados puedan ser en aplicaciones de modelos de aprendizaje automático para la detección de anomalías en transacciones con tarjetas de crédito y en futuras investigaciones relacionadas con este tema.

1.3 Planteamiento del problema.

El evidente crecimiento en el giro del negocio de la emisión de tarjetas de crédito y crecimiento en el volumen de datos generados en relación con las transacciones ejecutadas por los tarjeta habientes hacen que en consecuencia exista un incremento en la materialización de transacciones de fraude y en patrones de consumo inusuales, lo que a su vez ocasiona pérdidas económicas, se estima que para el año 2025 sumaran alrededor de $ 200 millones a nivel global. (Gestion, 2021) Esto ha generado una necesidad de abordar de manera más eficiente el tratamiento del fraude mediante el uso de técnicas matemáticas avanzadas con el objetivo de minimizar las pérdidas.

1.4 Objetivos

1.4.1 Objetivo general.

Desarrollar un Modelo de Clasificación basado en técnicas de aprendizaje profundo para identificar anomalías en transacciones ejecutadas con tarjetas de crédito enmarcado en la metodología MLOps.

1.4.2 Objetivos específicos.

Desarrollar un Modelo de clasificación que permita la identificación de transacciones fraudulentas y anomalías en los datos.

Realizar pruebas con el modelo para evaluar su precisión y proponer ajustes dentro del modelo para su refinamiento.

Desarrollar un conocimiento más profundo acerca del comportamiento de fraude generado con tarjetas de crédito y su funcionamiento.

CAPITULO II. FUNDAMENTACIÓN TEORICA

2. Marco teórico

2.1 ¿Qué es el fraude?

Según los autores del libro Fraud analytics using descriptive, predictive, and social network techniques: A guide to data science for fraud detection, Van Vlassealer y otros (Van Vlasselaer, 2015), describen al fraude en términos generales como un crimen poco común, bien considerado, imperceptiblemente oculto, que evoluciona con el tiempo, cuidadosamente organizado, y que aparece en muchas formas diferentes.

Según el diccionario de la RAE (RAE, n.d) fraude es la acción contraria a la verdad y a la rectitud, que perjudica a la persona contra quien se comete.

2.2 Fraude en tarjetas de crédito.

En el fraude con tarjetas de crédito, hay una toma no autorizada del crédito de otra persona. Algunos tipos comunes de fraudes con tarjetas de crédito incluyen la falsificación de tarjetas de crédito, el uso de tarjetas perdidas o robadas, o la adquisición fraudulenta de crédito a través de correo electrónico. (Van Vlasselaer, 2015)

En el Ecuador, según una investigación periodística del diario de Manabí, El Diario, existen diferentes maneras de fraude con tarjetas de crédito que generalmente cambian a través del tiempo. Se identifican dos categorías principales: los conocidos como fraudes de “tarjeta no presente” y los de “tarjeta presente”.

El primer caso se trata del tipo más común y ocurre cuando la información del titular de la cuenta de banco o la información contenida en el plástico de la tarjeta es robada y utilizada ilegalmente sin la presencia física de la tarjeta.

Esta estafa suele ocurrir en línea y puede ser el resultado de llamadas, correos electrónicos de phishing o suplantación de identidad, enviadas por estafadores que se presentan como empleados de instituciones financieras con el fin de robar información personal o financiera a través de llamadas telefónicas, enlaces o un programas maliciosos.

El segundo caso, ocurre cuando un vendedor pasa la tarjeta por un dispositivo que almacena su información y luego la utiliza para cargar otras compras que no realiza. (G., Mantuano, 2021)

2.3 Métodos actuales en la detección de fraudes.

Ante esta problemática los distintos actores del sistema financiero han considerado la utilización de técnicas avanzadas y de controles físicos sobre la información contenida en la tarjeta de crédito, la emisión de certificaciones para instituciones financieras que garanticen el proceso de venta segura, la emisión de claves temporales para cada transacción ejecutada, la implementación de la autenticación 3D Secure, la autenticación biométrica y las distintas campañas de concientización emitidas de manera masiva para generar conciencia en los usuario sobre el uso de la información sensible contenida en la tarjeta de crédito, son controles preventivo efectivos que permiten disminuir la materialización de transacciones fraudulentas, todo esto sumado al uso de modelos de Machine Learning de aprendizaje supervisado como los árboles de decisión, la regresión logística, las máquinas de soporte vectorial o K Vecinos cercanos o modelos basados en aprendizaje no supervisado como el clustering para agrupar tarjeta habientes en función de su comportamiento de consumo (Dornadula & Greetha, 2019) componen la realidad actual en la detección de transacciones fraudulentas generadas en tarjetas de crédito.

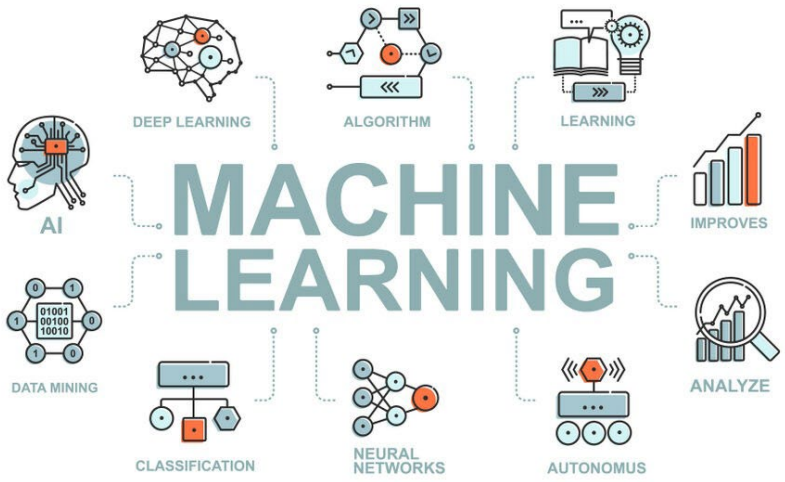
2.4 ¿Qué es el Machine Learning?

Pero ¿qué es el Machine Learning o aprendizaje automático? El Machine Learning es un campo científico y, más particularmente, una subcategoría de inteligencia artificial, que consiste en dejar que los algoritmos descubran «patterns», es decir, patrones recurrentes, en conjuntos de datos. Esos datos pueden ser números, palabras, imágenes, estadísticas, etc. (DataScientest, 2023)

El machine learning, el deep learning y las redes neuronales son subcampos de la inteligencia artificial. Sin embargo, las redes neuronales son en realidad un subcampo del machine learning, y el deep learning es un subcampo de las redes neuronales. (IBM, s.f)

En el campo del Machine Learning podemos identificar algunos tipos de algoritmos que se utilizan con frecuencia, principalmente y para motivos de esta investigación se definirán 3 clases, algoritmos de Aprendizaje Supervisado, algoritmos de Aprendizaje No Supervisado y algoritmos de Deep Learning.

Figura 1 Diagrama sobre el Machine Learning



(Metaphorce, 2022) Descubre todo lo que necesitas saber sobre el Machine Learning [Imagen]. Metaphorce

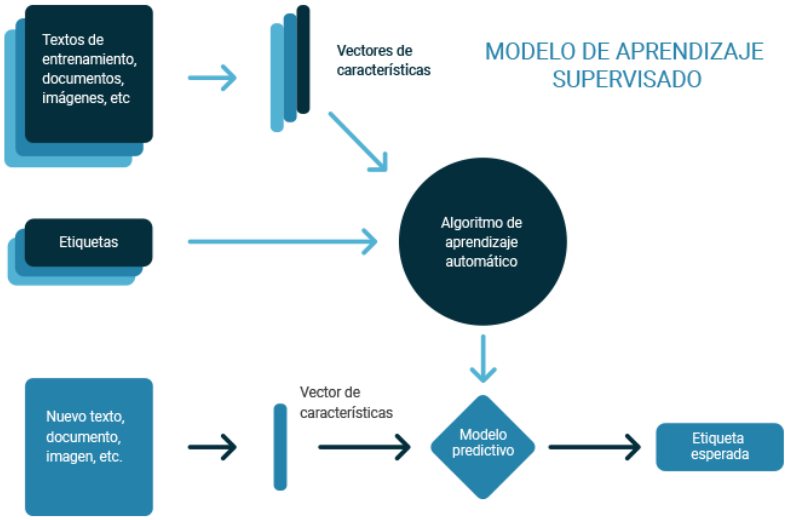
https://www.linkedin.com/pulse/descubre-todo-lo-que-necesitas-saber-sobre-el-machine-learning-/

2.4.1 Aprendizaje Supervisado

El aprendizaje supervisado es un campo del aprendizaje automático (Machine Learning) en el cual, partiendo de un conjunto de datos etiquetados, un algoritmo es capaz de entregar una predicción o una clasificación con precisión únicamente basado en los datos proporcionados.

Entre los principales algoritmos de aprendizaje supervisado tenemos la regresión lineal, la regresión logística, las máquinas de soporte vectorial, los árboles de decisión, entre otros.

Figura 2 Diagrama sobre el Aprendizaje Supervisado



(Gonzalez, 2018) Tipos de aprendizaje automático [Imagen], Meduim

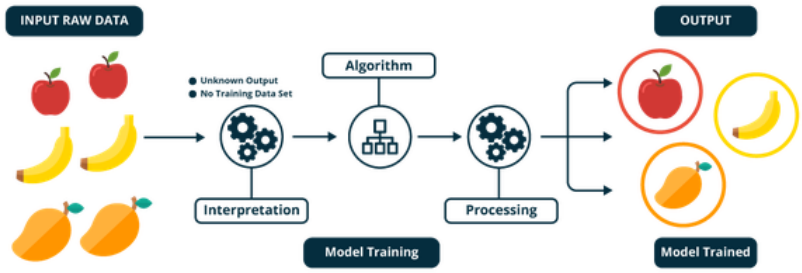
https://medium.com/soldai/tipos-de-aprendizaje-autom%C3%A1tico-6413e3c615e2

2.4.2 Aprendizaje No Supervisado.

Es una técnica parte del aprendizaje automático en el cual, a diferencia del aprendizaje supervisado, los datos de entrada no requieren estar etiquetados, los algoritmos se encargan de descubrir los patrones y relaciones existentes en la data por si solos, este tipo de algoritmos se

utilizan regularmente en la reducción de dimensiones, cuando la data tiene alta dimensionalidad, a través de PCA o Autoencoders y para generar agrupamientos de datos en función de sus características similares (Clustering).

Figura 3 Diagrama sobre el Aprendizaje No Supervisado



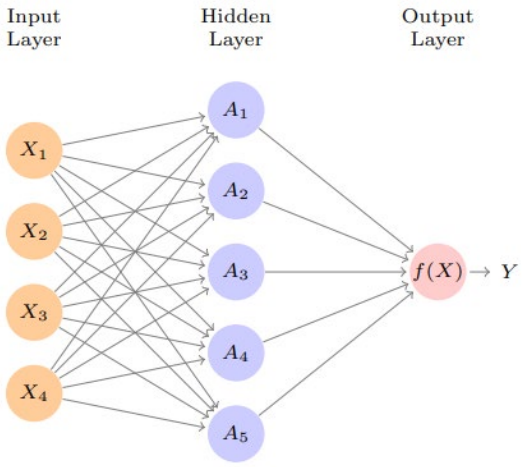
(Gonzalezi, 2019) Machine Learning: La base que tenes que tener [Imagen], Somospnt https://somospnt.com/blog/53-introduccion-a-machine-learning

2.4.3 Deep Learning.

El Deep Learning es una rama de Machine Learning compuesta por algoritmos que intentan imitar el aprendizaje humano y lograr resolver problemas de forma autónoma. (BBVA, 2023)

Los modelos de Deep Learning se basan en arquitecturas de redes neuronales inspiradas en el cerebro humano, estás, en su modelo más básico, constan de tres capas, la capa de entrada o input, las capas ocultas y las capas de salida u outputs, las neuronas situadas entre las capas de entrada se denominan capas ocultas. El término “Deep” (profundo) normalmente alude a la cantidad de capas ocultas de la red neuronal. Los modelos de Deep Learning pueden tener cientos o incluso miles de capas ocultas. (MATLAB & Simulink, s.f)

Figura 4 Composición básica de una Red Neuronal



(James, Witten, Hastie, & Tibshirani, 2013) An Introduction to Statistical Learning with Applications in R. [Imagen]

2.5 Redes Neuronales Artificiales.

Las redes neuronales artificiales son una abstracción del concepto de redes neuronales humanas, una neurona humana es una unidad funcional básica del sistema nervioso y permite transferir información, una neurona artificial es una unidad de cálculo básico que recibe una serie de entradas $x_1, x_2 \ldots, xn$ la red aplica una operación lineal seguida de una función de activación no lineal para obtener un output.

Matemáticamente se expresa de la siguiente manera.

$$y = \sigma\left(\sum_{i=1}^{n} \omega_i x_i + b\right)$$

Donde:

x_i: Entradas a la neurona.

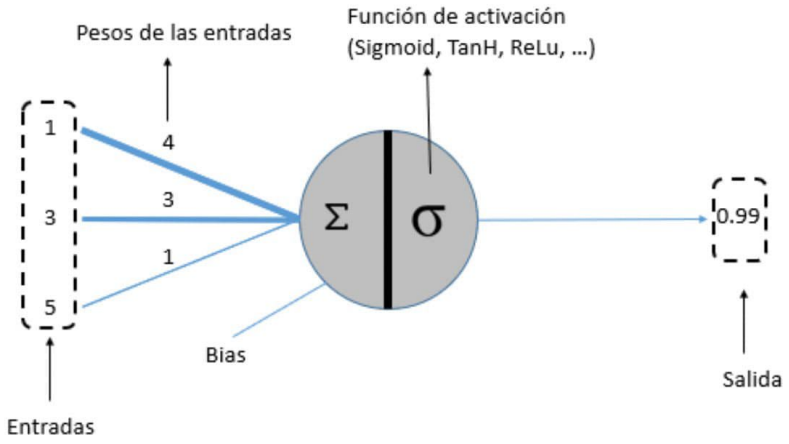
ω_i: Pesos asociados a cada entrada.

b: Sesgo (bias).

$\sum_{i=1}^{n} \omega_i x_i + b$: Combinación lineal de las entradas.

σ: Función de activación, que introduce no linealidad.

Figura 5 Estructura de una Neurona



(Xeridia, 2019) *Redes Neuronales artificiales: Qué son y cómo se entrenan.*

https://www.xeridia.com/blog/redes-neuronales-artificiales-que-son-y-como-se-entrenan-parte-i

2.5.1 Entrenamiento de una red neuronal.

Para el entrenamiento de una red neuronal se toman en cuenta algunos varios parámetros importantes como los hiperparametros, la arquitectura de la red y su utilidad, a continuación, se exponen algunos aspectos importantes de la estructura de las redes neuronales.

2.5.2 Propagación hacia Adelante

La propagación hacia adelante es el proceso de pasar las entradas a través de la red para obtener una salida. Los datos ingresan a través de la capa de entrada, pasan por las capas ocultas y se obtiene una salida a través de la capa de salida, esto se realiza de manera iterativa.
Matemáticamente se pude definir de la siguiente manera:

$$a^{(l+1)} = f\left(W^T \mathrm{x}^{(l)} + b^{(l)}\right)$$

En donde $a^{(l)}$ es la salida de la capa l, $W^{(T)}$ son los pesos y $b^{(l)}$ es el sesgo de la capa.

2.5.3 Funciones de activación.

Una función de activación es una función matemática que se aplica a la capa de salida de una neurona o a una capa de neuronas antes de pasar la señal a la siguiente capa. Su principal función es la de introducir no linealidad en el modelo, de esta manera la red neuronal aprende y representa relaciones complejas en los datos.

Dentro de las principales funciones de activación se pueden definir tres distintas, las cuales están adaptadas para resolver diferentes tipo de problemas.

Función Sigmoide

Toma cualquier valor de entrada y lo transforma en un valor entre 0 y 1. Es útil par problemas de clasificación.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Tangente Hiperbólica (TanH)

Toma cualquier valor de entrada y lo transforma en un valor de salida entre -1 y 1.

$$\mathrm{TanH}(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

ReLU

Toma cualquier valor de entrada, para valores negativos los vuelve cero en la salida, mientras que las entradas positivas permanecen sin cambios.

$$\mathrm{Re}\,L\,U(z) = \max(0, z)$$

2.5.4 Función de Pérdida (Loss Function)

Para entrenar una red neuronal, se define una función de costo que mide qué tan bien la red está haciendo su trabajo, este valor mide la diferencia entre el valor estimado y el valor real. Pueden ser pérdida logística, binary crossentropy o Log Loss.

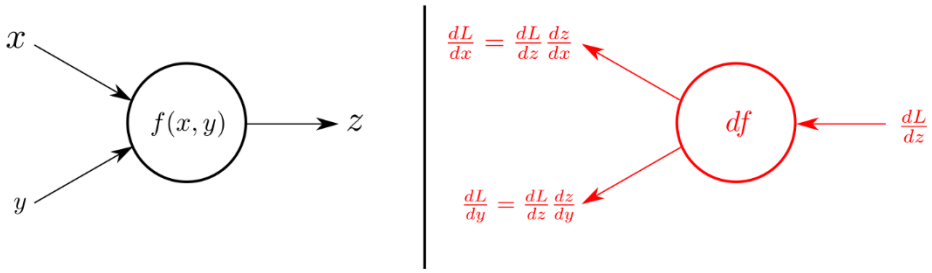
$$Perdida\ Logistica = -\frac{1}{m}\sum_{i=1}^{m}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

Donde m es el número de ejemplo de entrenamiento, y_i es el valor real en su i enésimo intento y $\hat{y}_i$ es el valor predicho.

2.5.5 Propagación hacia atrás (Backpropagation).

Durante el proceso de propagación hacia delante, las entradas pasan por cada una de las capas que componen la red neuronal para obtener una salida, el objetivo de la propagación hacia atrás, (Backpropagation) es que después de cada paso hacia adelante, el algoritmo realice un paso hacia atrás ajustando los pesos y sesgos del modelo esto con el objetivo de minimizar los errores en el proceso de aprendizaje y mejorar la precisión de las predicciones. Este proceso lo realiza calculando el gradiente de la función de costo y actualizando los parámetros usando el descenso del gradiente. (Unir, La Universidad de Internet, 2023)

Figura 6 Backpropagation



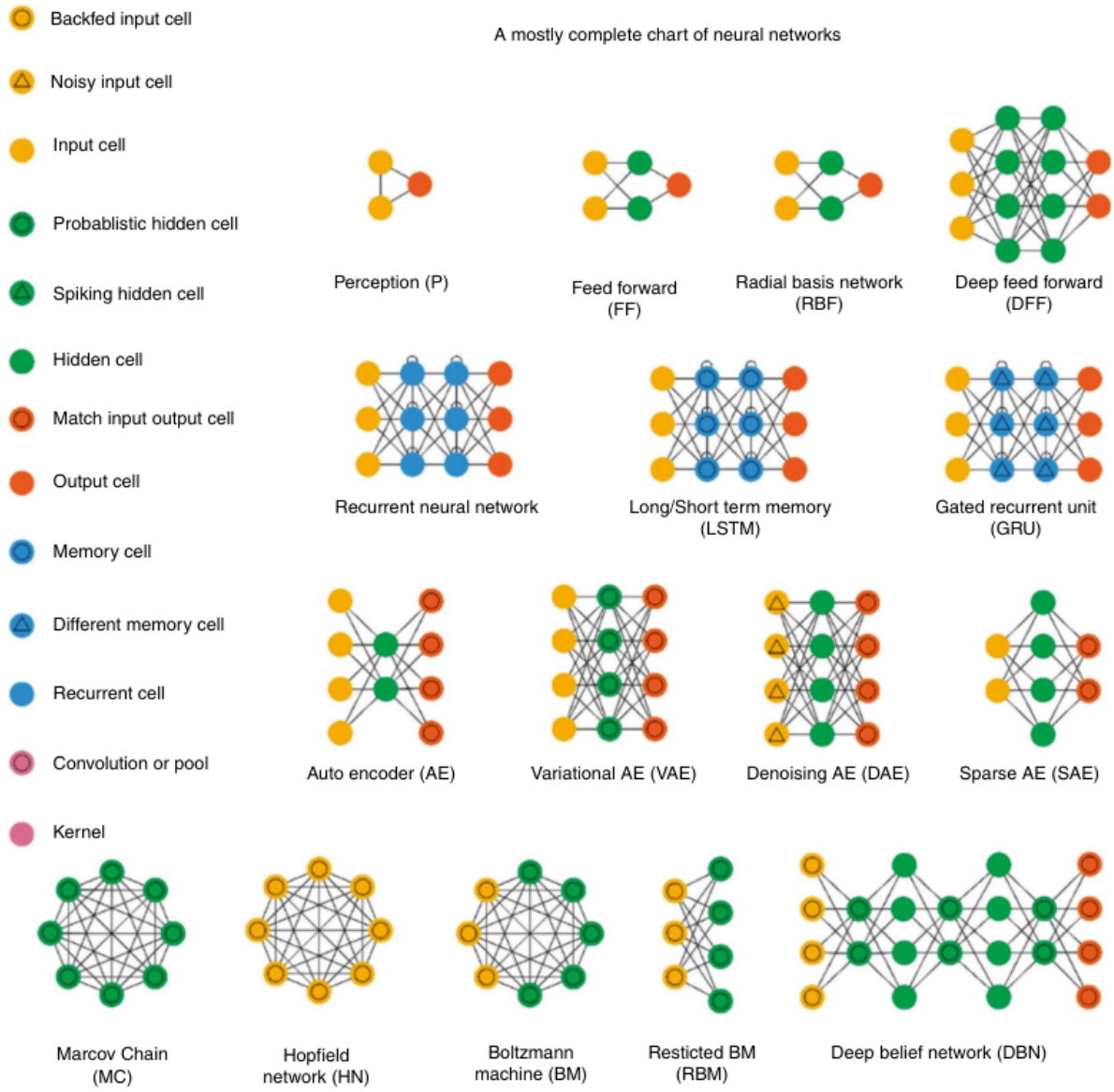
(Agarwal, s.f) Neural Network with BackPropagation. Implement a simple Neural network trained with backprogation in Python3. [Imagen] https://github.com/Vercaca/NN-Backpropagation

2.5.6 Arquitecturas de Redes Neuronales.

Existen diferentes arquitecturas de redes neuronales, cada una está diseñada para resolver distintos tipo de problemas, la arquitectura más simple está compuesta por tres capas: capa de entrada, capa oculta y capa de salida, estas se denominan redes neuronales artificiales, también existen las redes neuronales convolucionales cuyas características son el tener dos o más capas, las conexiones entre sus nodos no son directas, este tipo de redes funcionan para tareas de reducción de dimensiones o para la asociación en función de características comunes, tareas de aprendizaje no supervisado. Otra arquitectura también conocida son las redes neuronales recurrentes, que son bastante útiles para tareas de aprendizaje por refuerzo, análisis de series temporales y tareas de aprendizaje supervisado.

A continuación, se muestra un gráfico demostrativo de las diferentes arquitecturas de las redes neuronales.

Figura 7 Arquitecturas en Redes Neuronales



(Ahmad, et al., 2021) Artificial intelligence model and correlation for characterization and viscosity measurements of mono & hybrid nanofluids concerned graphene oxide/silica. [Imagen]

2.6 Autoencoders

Los Auto Encoders, son una arquitectura específica de las redes neuronales, se caracterizan por su uso en tareas de aprendizaje no supervisado, los autoencoders son eficientes en la reducción de dimensionalidad y en la extracción de características principales, así como en aprender representaciones eficientes de los datos y generar una reconstrucción de los datos sin perder sus características principales. (Cañadas, 2021)

2.6.1 Elementos de los autoencoders.

La arquitectura de un autoencoder está compuesta de tres partes: el codificador (Encoder), el espacio latente y el decodificador (Decoder)

- **Encoder**

Es el encargado de comprimir la información, representa la información de entrada en una dimensión menor.

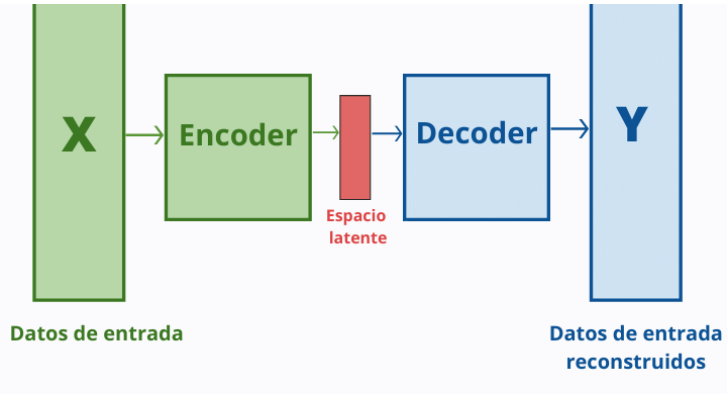
- **Espacio Latente**

Es la representación de los datos comprimidos. El autoencoder reconstruye las características a partir de aquí.

- **Decoder**

Reconstruye la entrada original a una dimensión similar a la entrada.

Figura 8 Arquitectura del Autoencoder



(Cañadas, 2021) Que son, Arquitectura Y sus Aplicaciones. [Imagen] Ab Datum. https://abdatum.com/tecnologia/autoencoders

2.6.2 Función de Perdida en Autoencoders

A diferencia de la función de perdida de una red neuronal tradicional, en un autoencoder la función de perdida trata de minimizar la diferencia entre la entrada y la salida reconstruida. Regularmente se utiliza el error cuadrático medio (MSE)

$$\mathcal{L}(x, \hat{x}) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2$$

En donde n es el número de ejemplos en el conjunto de datos.

2.6.3 Autoencoders en la prevención de fraudes.

Empresas como PayPal han adaptado distintos modelos de Deep Learning para detectar fraudes en su plataformas de pago, en su presentación en noviembre de 2018 en Santa Clara, Estados Unidos, en la AI & Big Data Expo North América (Sharma, 2018), realizaron una revisión general respecto al panorama y cuáles eran sus estrategias para combatir el fraude, entre los principales métodos

expusieron están las redes neuronales recurrentes, los modelos basados en redes Long Short Term Memory, también expusieron métodos de investigación basados en aprendizaje no supervisado en el cual se destaca el uso Las Máquinas de Boltzmann Restringidas, el aprendizaje por transferencia y el uso de Autoencoders.

2.7 Metodología MLOps

Las operaciones de aprendizaje automático (MLOps) son un conjunto de prácticas para los flujos de trabajo cuyo objetivo es optimizar el proceso de implementación y mantenimiento de los modelos de aprendizaje automático. (RedHat, 2023)

Las operaciones de aprendizaje automático (MLOps) nacen de DevOps, la cual es una metodología de trabajo ágil que busca incrementar la capacidad de una organización en el desarrollo de software y administración de su infraestructura, al integrar a su cultura diferentes características como el trabajo del equipo durante todo el ciclo de vida de la proyecto, desde el desarrollo, la realización de pruebas, la implementación y el despliegue de operaciones, también busca la integración de los equipos de desarrollo y de operaciones de TI.

2.7.1 Principios de DevOps.

Los principios DevOps están compuestos de dos pilares fundamentales: los principios CI/CD y la implementación continua.

2.7.1.1 Principios CI/CD

La integración continua CI (Continuous Integration) es la practica en la cual los desarrolladores fusionan el código en un repositorio común, esto les permite la realización de pruebas para detectar y resolver errores.

La entrega continua CD (Continuous Delivery) hace referencia a la automatización del proceso de entrega de software desde la integración del código, pasando por la etapa de pruebas y hasta llegar a la entrega al usuario final. Esta parte del proceso tiene intervención humana, los desarrolladores deciden cuando se entrega el software. (ILIMIT, 2020)

2.7.1.2 Implementación continua.

Está estrechamente relacionada con la entrega continua (CD), se encarga de automatizar el proceso de entrega del software eliminando la acción humana, sigue una serie de pasos que deben ejecutarse en orden y de forma correcta, se basa en microservicios.

Estos principios establecen la base de la metodología de trabajo MLOps que pretende integrar las operaciones de Machine Learning al desarrollo de software garantizando el entrenamiento, actualización, implementación y monitoreo al mismo tiempo que tiene en cuenta la seguridad, escalabilidad y el despliegue continuo de las operaciones de ML. (ILIMIT, 2020)

2.7.2 Etapas de la metodología MLOps.

El flujo de trabajo en MLOps se puede definir en siete etapas principales dentro de las cuales se considera todo el proceso desde la creación, el modelado, el entrenamiento, el despliegue en producción y el monitoreo de un modelo de aprendizaje automatizado, comienza por la fase de entendimiento del problema del negocio y del entendimiento los requerimientos del modelo a desarrollar. (Medeiros, 2023)

- **Análisis de los requerimientos del modelo.**

En esta etapa se definen los requerimientos del modelo, se plantea el problema del negocio y se analiza el valor agregado que un modelo de Machine Learning puede aportar para la resolución de un problema específico, también se analiza el valor agregado relacionado al capital humano y como este aporta a la resolución del problema antes de aplicar métodos de aprendizaje automático.

Se determinan las funciones necesarias, los objetivos, las herramientas, las tareas prioritarias y todo lo que involucre a los diferentes stakeholders.

- **Recolección y preparación de datos.**

En esta etapa se direcciona la recolección de los datos, los procesos de ETL, (extracción, transformación y carga), el abordaje que dé se debe dar respecto a los datos que se van a utilizar, desde su obtención, pasando por su limpieza, emparejamiento, hasta llegar al análisis exploratorio de datos (EDA), esto con el objetivo de garantizar la calidad en los datos con los que van a ser entrenados los modelos y poder obtener insigths valiosos que ayuden a resolver el problema del negocio planteado.

- **Ingeniería de características.**

Una vez obtenido datos limpios y una conocimiento base generado por el análisis exploratorio, conviene trabajar y especificar de qué manera el modelo de aprendizaje automático va a recibir las características (features) para ser entrenado, esto implica realizar una normalización en la data, la imputación de las características, la creación de nuevas características a partir de las interacciones entre los features, hasta utilizar técnicas de reducción de dimensionalidad, estandarización y normalización de los datos.

- **Entrenamiento del modelo.**

En esta etapa se consideran los diferentes modelos en la resolución del problema del negocio, se establece el modelo correcto para resolver el problema específico planteado, en esta etapa se realiza el ajuste de los hiperparámetros del modelo, la construcción del modelo, su entrenamiento y su refinamiento.

- **Evaluación en modelo.**

Durante la evaluación del modelo se obtienen las distintas métricas para medir la precisión de un modelo de Machine Learning, algunas métricas que se utilizan pueden ser la precisión, la efectividad, F1 Score, la curva ROC, se toma en cuenta tanto el valor generado por la ejecución del modelo, así como la del valor que genera al giro del negocio.

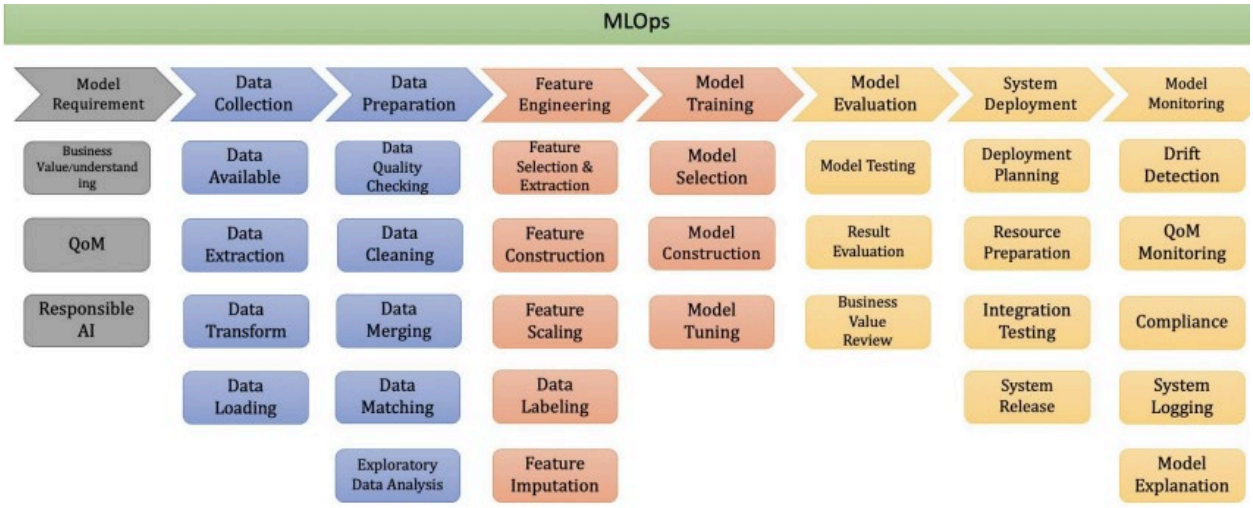
- **Despliegue.**

Durante la etapa de despliegue se considera la manera en que se va a lanzar a producción un modelo de aprendizaje automático, toma en consideración plataformas o sistemas integrados que garanticen el proceso desde el testeo hasta la entrega al usuario, en esta etapa también se pueden definir Pipelines que ayuden en la automatización de procesos repetitivos durante el proceso de despliegue a producción.

- **Monitoreo del modelo.**

Un elemento clave dentro de MLOps, es el monitoreo continuo que ofrece sobre los modelos ya implementados en producción, esto se debe a su filosofía basada en los principios CI/CD, también se considera el mantenimiento en el largo plazo, el control de versiones, la calidad del modelo y la importancia de mantener en el tiempo modelos inteligentes que sean explicables.

Figura 9 Etapas de desarrollo en MLOps



(Medeiros, 2023) MLOps spanning whole machine learning life cycle: Paper summary. [Imagen] MarkTechPost

https://www.marktechpost.com/2023/07/28/mlops-spanning-whole-machine-learning-life-cycle-paper-summary/

2.8 Fundamentos de programación en Python.

2.8.1 ¿Qué es Pyhton?

Python es un lenguaje de alto nivel de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código. Se trata de un lenguaje de programación multiparadigma, ya que soporta parcialmente la orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma. (Wikipedia, s.f)

Figura 10 Logo de Python

(Paz, 2023) Primeros pasos con Python: ¡Hola, mundo! [Imagen]

https://blog.vermiip.es/primeros-pasos-con-python-hola-mundo/

2.8.2 Python para la Ciencia de Datos.

Python ofrece varias características que lo vuelven atractivo para utilizarlo como lenguaje de programación en tareas de Machine Learning, su sintaxis fácil de leer y entender, el soporte de su comunidad y las librerías diseñadas específicamente para la gestión de datos, procesamiento analítico y la creación de modelos de aprendizaje automático lo convierten en una herramienta útil para de la Ciencia de Datos.

2.8.3 Principales librerías y frameworks.

Dentro de las principales librerías que tiene Python para el manejo de datos en ciencia de datos tenemos las librerías Pandas, Numpy, Matplotlib, Sckit-Learn y Keras/Tensorflow. (Verne Academy, 2022)

- **Pandas.**

Facilita el manejo de datos, permite leer distintos tipos de archivos o bases de datos, extensiones tipo csv, htlm, xls, hdfs, entre otros.

- **Numpy**

Facilita el manejo de operaciones de matrices, permite crear todo tipo de estructuras numéricas de diferentes dimensiones, permite realizar operaciones aritméticas entre matrices.

- **Mathplotlib**

Es una librería que permite la creación estática, animada e interactiva de visualizaciones en Python.

- **Sckit-Learn**

Contiene una gran variedad de algoritmos optimizados para tareas de machine Learning, dentro de su basto catalogo se destacan algoritmos de regresión, clasificación, reducción de dimensiones, clustering, entre otros.

- **Keras/Tensorflow.**

Esta Liberia contiene variedad de algoritmos de aprendizaje profundo, algoritmos optimizados para redes neuronales, entre las más destacadas están los Autoencoders, las redes neuronales convolucionales, redes LSTM y muchas más.

2.8.4 Código fuente.

A continuación, se exponen algunas líneas de código que pretenden ilustrar la sintaxis del Python. Se pueden realizar operaciones básicas

```
x = 2
x = x + 1
x
3
```

Se importan las librerías

```
# Importar las librerias
import numpy as np
```

Utilización de la función random de la librería numpy.

```
import numpy as np
data = np.random.randn(1,2)
data
array([[ 1.17376772, -0.78566716]])
```

Creación de arreglos (Arrays)

```
data1 = [1,2.5,3,9,4]
arr1 = np.array(data1)
arr1
array([1. , 2.5, 3. , 9. , 4. ])
```

Otra de las funciones principales es la de leer distintos tipos de archivos, esto se lo realiza con la librería pandas.

```
# Importar la libreria
import pandas as pd
```

Se crea un data frame con la información del archivo .csv

```
df = pd.read_csv('C:\\Users\\clinc\\Desktop\\PUCE\\sample_submission.csv')
df
```

CAPITULO III. RECOPILACIÓN Y PREPROCESAMIENTO DE DATOS

3.1 Selección del conjunto de datos representativos.

Para el desarrollo de este trabajo se utilizará un dataset obtenido de la plataforma Kaggle.com que contiene la información de transacciones presentes generadas por tarjetahabientes europeos en Septiembre del 2013.

El dataset contiene transacciones que ocurrieron en dos días, con un total de 284,807 observaciones de las cuales 492 son transacciones fraudulentas. El dataset está altamente desbalanceado.

El dataset está compuesto principalmente por 30 variables predictoras y 1 variable respuesta, contiene la variable Time, que es tiempo transcurrido en segundos desde la primera transacción generada, contiene la variable Amount, que es el monto de la transacción, contienen las variables V1, V2, V3… hasta la variable V28, que son el resultado de un análisis de componentes principales (PCA) y también contiene la variable respuesta Class que es binaria y explica que transacción es fraudulenta o no fraudulenta.

0 = No fraude

1 = Fraude.

El dataset está compuesto por las siguientes variables.

Time: El tiempo transcurrido en segundos desde la primera transacción.

Amount: El monto de la transacción

Class: Variable objetivo que define que transacción es de fraude o no fraude.

Variables resultantes de la aplicación de del PCA van desde la V1, V2, V3, … hasta la V28

3.2. Preprocesamiento.

El primer paso es importar las librerías que vamos a utilizar durante esta etapa de preprocesamiento de los datos, las importamos con las siguientes líneas de código.

```
# Importar las librerias
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Leemos el archivo creditcard.csv.

```
# Leemos el archivo
path = 'C:\\Users\\clinc\\Desktop\Recursos proyecto\\creditcard.csv'
transacciones = pd.read_csv(path)
transacciones
```

Figura 11 Dataset Cargado

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.06692
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.33984
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.68928
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.17557
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.14126
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480	-0.50934
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012463	-1.01622
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037501	0.64013
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163298	0.12320
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376777	0.00879

284807 rows × 31 columns

Gráfico que muestra la carga del dataset en el ambiente de VSC, autoría propia, 2024.

El dataset tiene 284.807 observaciones y registra 31 columnas, 30 variables de predictoras y 1 variable respuesta.

Guardaremos el dataset como una variable dentro del ambiente de Visual Studio Code con el nombre **transacciones.**

Realizamos una exploración inicial de los datos para verificar que información tenemos, lo realizamos con las siguientes líneas de código.

```
# Imprimimos las primeras 10 linenas del conjunto de datos
transacciones.head(10)

# Imprimimos los ultimas 10 lineas de los datos
transacciones.tail(10)
```

Verificamos el tipo de datos que componen el dataset. Comprobamos que las variables mantienen la clase correspondiente para nuestro análisis. La variables Time, Amount y las resultantes del PCA V1, V2, V3, … V28 son variables cuantitativas continuas y pueden tomar cualquier valor numérico dentro de un rango. La variable Class es una variable cuantitativa discreta que por su naturaleza solo puede tomar un conjunto específico de valores, en este caso 0 y 1.

```
# Vemos el tipo de datos que tenemos
transacciones.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 32 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   Time        284807 non-null  float64
 1   V1          284807 non-null  float64
 2   V2          284807 non-null  float64
 3   V3          284807 non-null  float64
 4   V4          284807 non-null  float64
 5   V5          284807 non-null  float64
 6   V6          284807 non-null  float64
 7   V7          284807 non-null  float64
 8   V8          284807 non-null  float64
 9   V9          284807 non-null  float64
 10  V10         284807 non-null  float64
 11  V11         284807 non-null  float64
 12  V12         284807 non-null  float64
 13  V13         284807 non-null  float64
 14  V14         284807 non-null  float64
 15  V15         284807 non-null  float64
 16  V16         284807 non-null  float64
 17  V17         284807 non-null  float64
 18  V18         284807 non-null  float64
```

```
 19  V19         284807 non-null  float64
 20  V20         284807 non-null  float64
 21  V21         284807 non-null  float64
 22  V22         284807 non-null  float64
 23  V23         284807 non-null  float64
 24  V24         284807 non-null  float64
 25  V25         284807 non-null  float64
 26  V26         284807 non-null  float64
 27  V27         284807 non-null  float64
 28  V28         284807 non-null  float64
 29  Amount      284807 non-null  float64
 30  Class       284807 non-null  int64
 31  TimeHoras   284807 non-null  float64
dtypes: float64(31), int64(1)
memory usage: 69.5 MB
```

Se debe verificar si existen valores NaN o valores nulos y perdidos, lo realizamos con las siguientes líneas de código.

```
# Validamos si existen valores nulos o perdidos
transacciones.isnull().sum()

Time         0
V1           0
V2           0
V3           0
V4           0
V5           0
V6           0
V7           0
V8           0
V9           0
V10          0
V11          0
V12          0
V13          0
V14          0
V15          0
V16          0
V17          0
V18          0
V19          0
V20          0
V21          0
V22          0
```

```
V23           0
V24           0
V25           0
V26           0
V27           0
V28           0
Amount        0
Class         0
TimeHoras     0
dtype: int64
```

El dataset está completo, no mantiene valores perdidos o valores nulos, no es necesario generar una imputación de la data, ya sea a través de la media, la mediana y algún otro valor, como el valor que más se repita.

Ahora vamos a validar si existen valores 0, específicamente en las variables Time y Amount.

```
# Calculo para valores 0 en la variable Time
ceros_horas = (transacciones['Time'] == 0).sum()
ceros_horas
2


# Calculo para valores 0 en la variable Amount(Monto)
ceros_amount = (transacciones['Amount'] == 0).sum()
ceros_amount
1825
```

La variable Time mantiene dos valores 0 pero no son significativas debido que el tamaño total de las observaciones del dataset es de 284.807 por lo que no es necesario generar la eliminación de estos dos registros a través de un método de drop().

Respecto a la variable Amount, en el análisis de transacciones generadas con tarjetas de crédito, es normal encontrar valores aprobados por el monto de $ 0.00 debido que estos valores regularmente corresponden al registro de los datos de la tarjeta en plataformas de pago a través de internet o transacciones de prueba cuando la transacción es tarjeta presente, el comercio en donde se realiza el consumo carga a la cuenta del tarjeta habiente este valor, no se aprueba ningún valor, pero los datos quedan registrados en la plataforma o para comprobar que la tarjeta se encuentra

activa para generar consumos. Este valor por sí solo no indica si una transacción es o no fraudulenta.

Por último, vamos a crear una nueva variable a partir de la variable Time ya que se encuentra expresada en segundos, la vamos a transformar para disminuir la escala y que este expresada en horas, lo realizamos con las siguientes líneas de código.

```
# La variable Time esta expresada en segundos transcurridos desde la
primera transaccion, la vamos a transformar a
# horas transcurridas, esto lo conseguimos dividiendo el valor de cada
observacion en Time para 3600 segundos que son los segundos
# contienen una hora. Vamos a crear una nueva variable llamada
TimeHoras
# TimeHoras = Time[n]/3600

transacciones['TimeHoras'] = transacciones['Time']/3600
```

Figura 12 Creación de la Variable TimeHoras

V16	V17	V18	V19	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class	TimeHoras
0401	0.207971	0.025791	0.403993	0.251412	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0	0.000000
3917	-0.114805	-0.183361	-0.145783	-0.069083	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0	0.000000
0083	1.109969	-0.121359	-2.261857	0.524980	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0	0.000278
9647	-0.684093	1.965775	-1.232622	-0.208038	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0	0.000278
1449	-0.237033	-0.038195	0.803487	0.408542	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0	0.000556
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7641	1.991691	0.510632	-0.682920	1.475829	0.213454	0.111864	1.014480	-0.509348	1.436807	0.250034	0.943651	0.823731	0.77	0	47.996111
1757	-0.025693	-1.221179	-1.545556	0.059616	0.214205	0.924384	0.012463	-1.016226	-0.606624	-0.395255	0.068472	-0.053527	24.79	0	47.996389
0716	0.313502	0.395652	-0.577252	0.001396	0.232045	0.578229	-0.037501	0.640134	0.265745	-0.087371	0.004455	-0.026561	67.88	0	47.996667
8577	0.509928	1.113981	2.897849	0.127434	0.265245	0.800049	-0.163298	0.123205	-0.569159	0.546668	0.108821	0.104533	10.00	0	47.996667
2620	-0.660377	0.167430	-0.256117	0.382948	0.261057	0.643078	0.376777	0.008797	-0.473649	-0.818267	-0.002415	0.013649	217.00	0	47.997778

Gráfico que muestra la variable creada TimeHoras, autoría propia, 2024.

3.3 Análisis Exploratorio de los Datos (EDA)

3.3.1 Primeros estadísticos.

Empezamos realizando un análisis visual de la distribución de las frecuencias de las clases, para eso creamos un gráfico de barras con las siguientes líneas de código.

```
# Agrupamos en función de la variable que queremos predecir, en este
caso la variable Clase(Class) que contiene si las transacciones
# son de fraudulentas o no fraudulentas.

# 0 = No Fraude
# 1 = Fraude


transacciones.groupby('Class').size()
Class
0    284315
1       492
dtype: int64
```

Creamos un gráfico para realizar la interpretación visual.

```
# Generamos un gráfico para entenderlo mejor de manera visualmente
# Creamos el gráfico
fraud_counts = transacciones.groupby('Class').size()

# Datos del grafico
labels = fraud_counts.index
counts = fraud_counts.values

# Disenamos el grafico
plt.figure(figsize=(8,6))
plt.bar(labels, counts, color=['navy', 'red'])

# Etiquetas
plt.title('Transacciones fraudulentas Vs Transacciones no
Fraudulentas')
plt.xlabel('Tipo de transaciones')
plt.ylabel('Numero de transacciones')
plt.xticks(labels, ['No Fraude', 'Fraude'])

# Imprimir el resultado
plt.show()
```

Figura 13 Diagrama de Barras Fradue / No Fraude

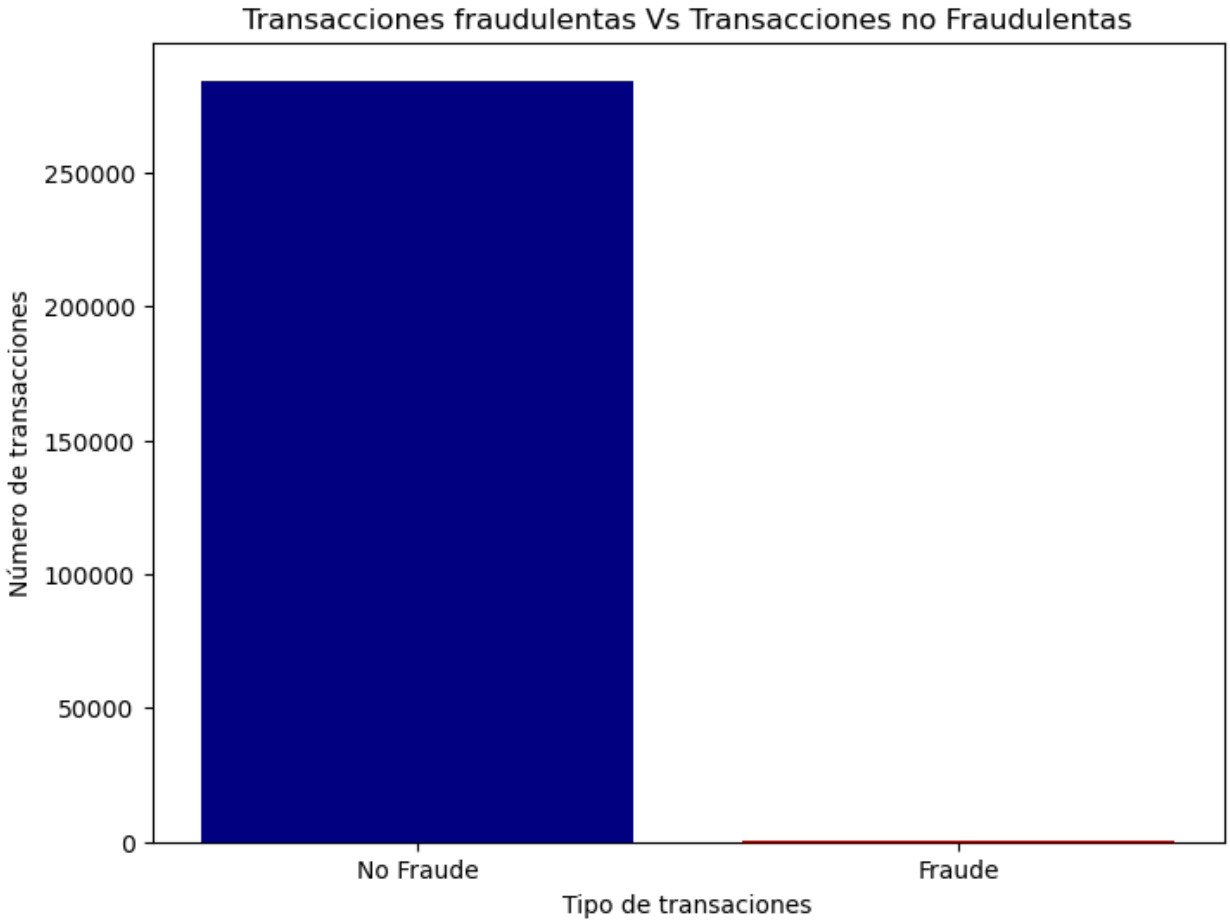


Gráfico de barras del Fraude y No Fraude, autoría propia, 2024.

Se puede observar que existe un claro desbalance entre las clases, las observaciones que pertenecen a la clase No Fraude son 284.315 mientras que las observaciones que pertenecen a la clase Fraude son 492 y representan apenas el 0,0017% del total de las observaciones.

Ahora vamos a representar la relación entre las variables TimeHoras y la variable Amount a través de una dispersión.

Figura 14 Dispersión transacciones totales

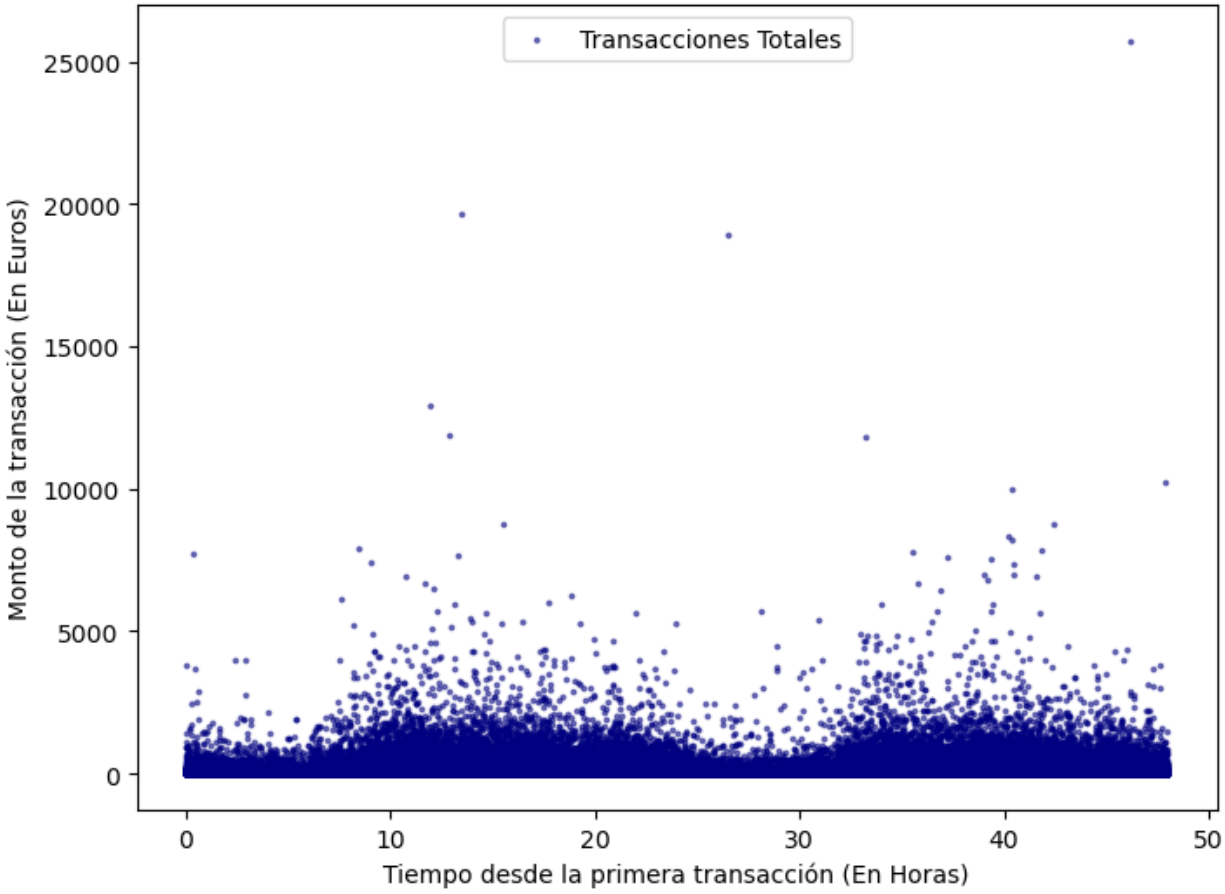


Gráfico de dispersión transacciones normales, autoría propia, 2024.

Se puede observar que principalmente existen dos grupos de puntos en donde están más acumuladas las transacciones, una nube de puntos agrupada entre las 10 y 25 horas y otro grupo de puntos agrupado entre 30 y 45 horas, esto será explorado más adelante cuando se analice la variable TimeHoras y su distribución de probabilidad. Respecto al monto existen valores atípicos pero la mayoría de los puntos se agrupa desde el valor de $ 0.00 hasta los $ 7000.

Resaltamos los puntos que corresponden a las transacciones de fraude.

Figura 15 Disperción transacciones de fraude y no fraude

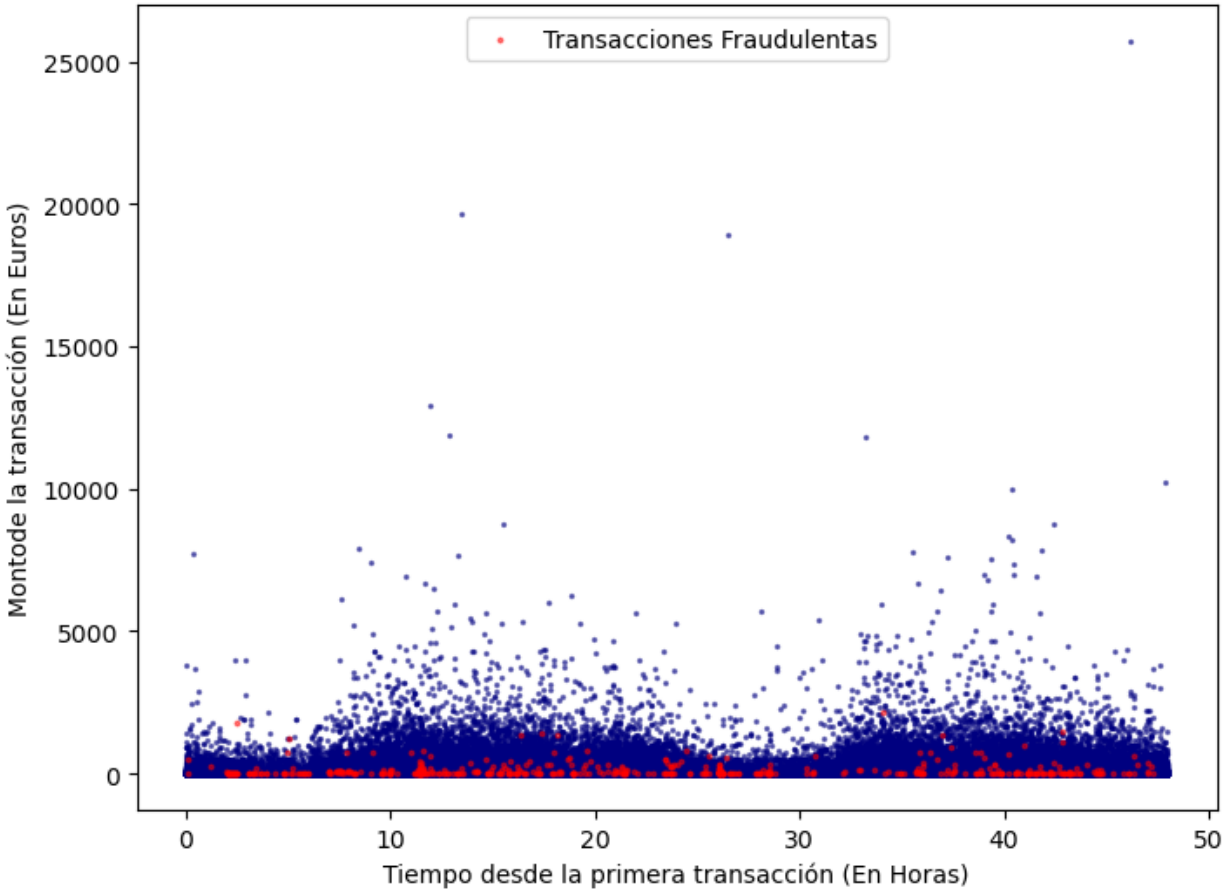


Gráfico de dispersión transacciones de fraude y no fraude, autoría propia, 2024.

Aparentemente, no se observa un patrón en particular que sigan las transacciones de fraude respecto a las transacciones normales, probablemente existan más acumulación de transacciones fraude en los grupos detectados anteriormente, pero se debe realizar pruebas para obtener una conclusión válida.

Revisamos solo las observaciones que corresponden a las transacciones de fraude.

Figura 16 Dispersión Transaciones de Fraude

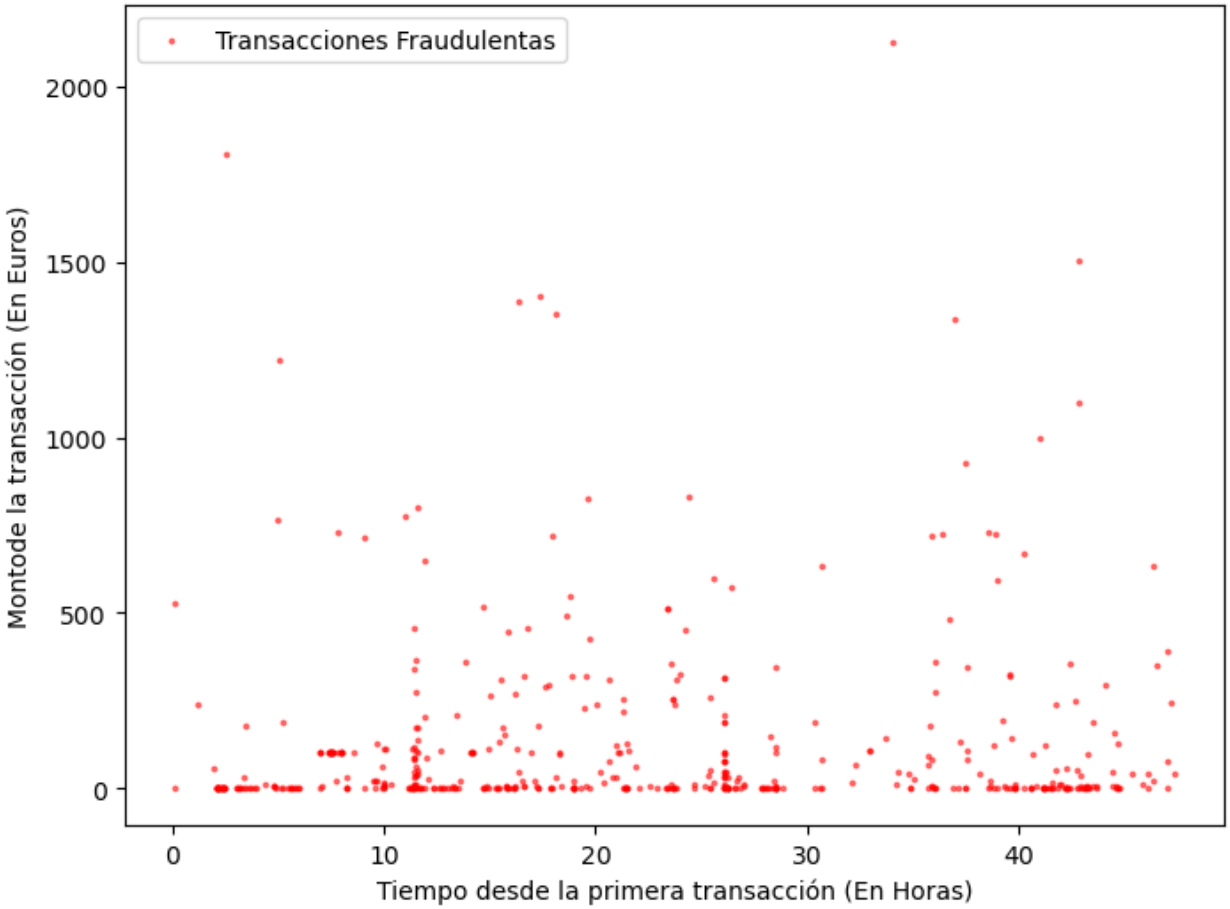


Gráfico de dispersión transacciones de fraude, autoría propia, 2024.

Se puede observar que las transacciones de fraude hacienden hasta un monto mayor a $ 2000 pero no alcanzan montos superiores a $ 25.000 como las transacciones normales, es decir las transacciones de fraude están agrupadas en montos menores a $ 2500. De la misma manera, aparentemente parece ser que existe más cantidad de transacciones entre los rangos de 10 a 25 horas y de 35 a 45 horas.

- **Variable TimeHoras**

Se imprimen los principales estadísticos de la variable TimeHoras.

```
# Imprimimos los estadisticos basicos de la varibale time.
transacciones['TimeHoras'].describe()
```

```
count    284807.000000
mean         26.337183
std          13.191152
min           0.000000
25%          15.055972
50%          23.525556
75%          38.700139
max          47.997778
Name: TimeHoras, dtype: float64
```

Existen 284.807 observaciones, con una media de tiempo en horas desde la primera transacción de 26,33 horas, la desviación estándar es de 13,19 horas lo que implicaría que los tiempos en las transacciones están bastante dispersos respecto a la media, el tiempo mínimo es de 0 horas mientras que el máximo es de 48 horas. El primer cuartil nos indica que el 25% de las transacciones ocurrieron dentro de las primeras 15 horas, el 50% ocurrieron antes de las 23 horas, el 75% de las transacciones ocurrieron antes de 38 horas.

Generamos una gráfica para verificar la distribución y la frecuencia.

```
# Subplot 1: Histograma y gráfico de densidad
plt.subplot(2, 1, 1)
sns.histplot(transacciones['TimeHoras'], color='orange', kde=True)
plt.title('Histograma y Gráfico de Densidad Variable TimeHoras')
plt.xlabel('Horas')
plt.ylabel('Frecuencia')
plt.grid(linewidth=0.2)

# Subplot 2: Boxplot
plt.subplot(2, 1, 2)
sns.boxplot(x=transacciones['TimeHoras'])
plt.title('Boxplot de la Variable TimeHoras')
plt.xlabel('Horas')
plt.grid(linewidth=0.2)
plt.tight_layout()
plt.show()
```

Figura 17 Histograma y Boxplot TimeHoras

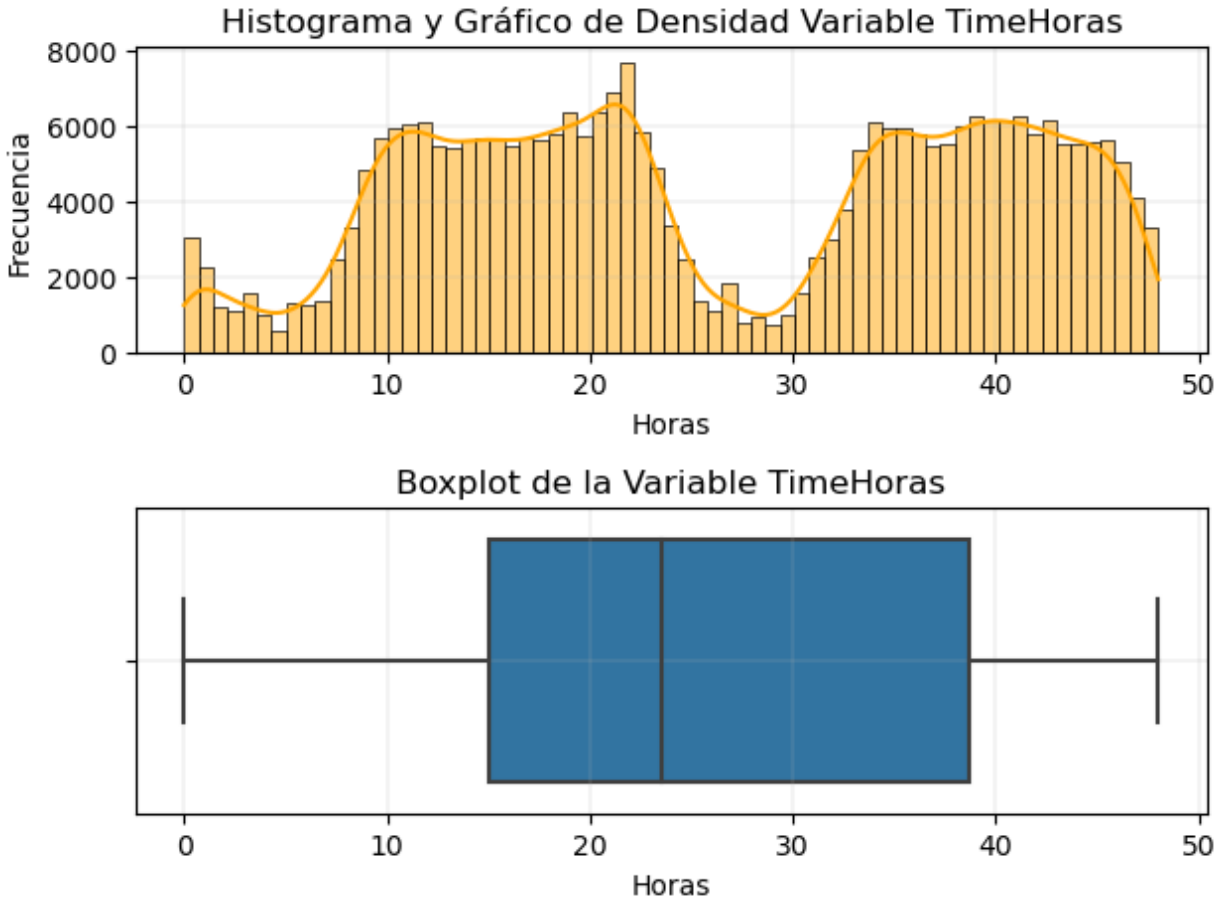


Gráfico Histograma y Boxplot TimeHoras, autoría propia, 2024.

A través del Histograma se puede verificar que los datos de la variable TimeHoras siguen una distribución Bimodal, estos significan que existen dos picos o modos distintos en la distribución de los tiempos de las transacciones, esto puede sugerir que las transacciones tienden a agruparse en dos periodos de tiempo diferentes. El grafico de boxplot trata de aproximar la distribución de los datos a una distribución normal, pero dado que los datos siguen una distribución bimodal no resulta de mucha utilidad para este caso

- **Variable Amount**

Se imprimen los principales estadísticos de la variable Amount.

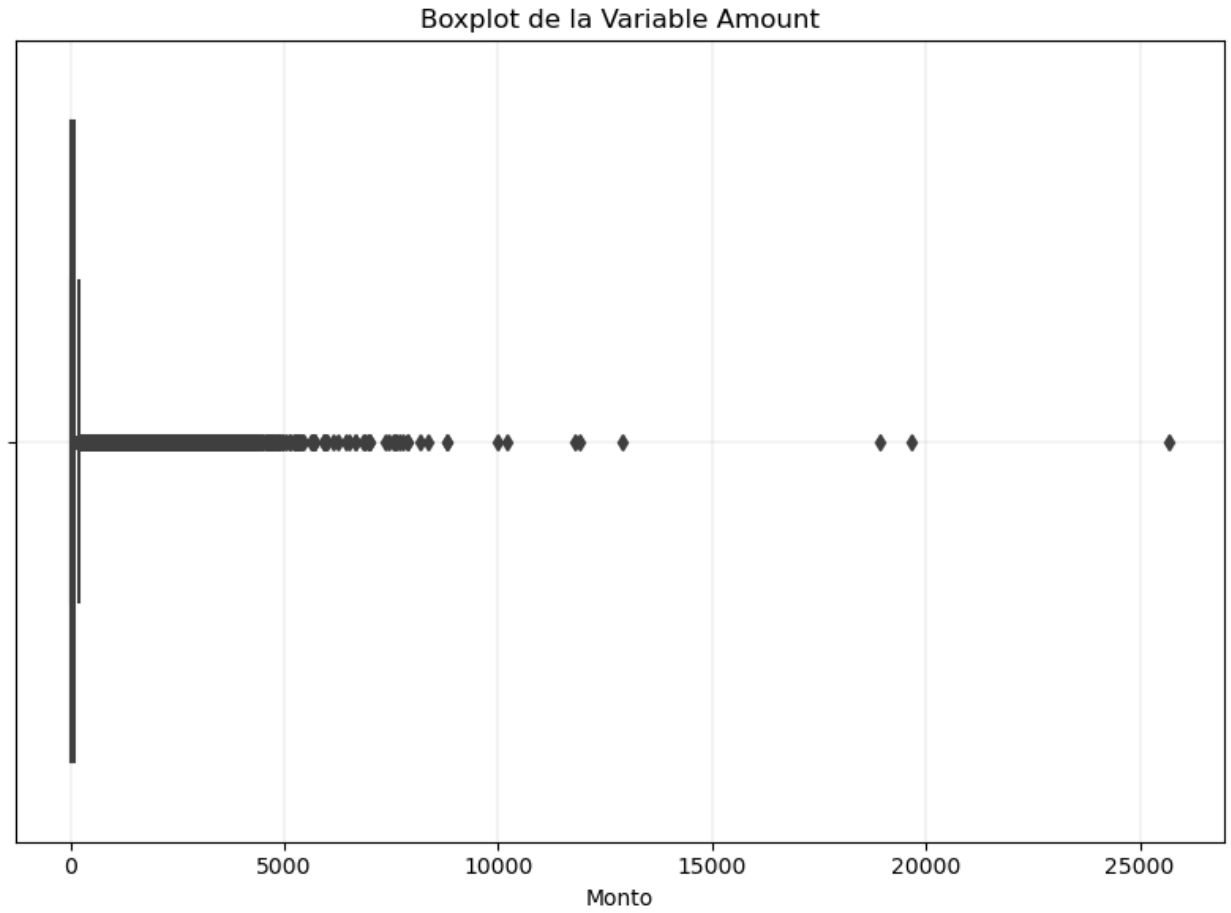
```
transacciones['Amount'].describe()
count    284807.000000
mean         88.349619
std         250.120109
min           0.000000
25%           5.600000
50%          22.000000
75%          77.165000
max       25691.160000
Name: Amount, dtype: float64
```

El número de observaciones es de 284807, la media del monto de las transacciones es de $ 88,34 y mantiene una desviación estándar del $ 250,00 lo que indicaría una altísima dispersión del monto de las transacciones respecto a la media, esto podría sugerir que los datos no siguen una distribución normal, el monto mínimo del monto de una transacción es de $ 0.00 y el monto máximo es de $ 25.691,16. El primer cuartil indica que el 25% de las transacciones tienen un monto de $ 5,60, el 50% tienen un monto de hasta $ 22, el 75% de las transacciones tienen hasta $ 77,17.

Generamos el gráfico correspondiente.

```
# Gráfico Boxplot Variable Amount
plt.figure(figsize=(8,6))
sns.boxplot(x=transacciones['Amount'])
plt.title('Boxplot de la Variable Amount')
plt.xlabel('Monto')
plt.grid(linewidth=0.2)
plt.tight_layout()
plt.show()
```

Figura 18 Boxplot Variable Amount



Boxplot Variable Amount, autoría propia, 2024.

Se puede observar que los datos podrían no seguir una distribución normal, registra varios valores atípicos, también se observa que la mayoría de las observaciones se agrupan en el valor de 0 que por la escala no es posible verificar a que valor corresponden, se puede sugerir realizar una transformación logarítmica para ajustar la distribución de la variable y obtener valores más normales.

3.3.2 Pruebas de Hipótesis.

Vamos a realizar pruebas de hipótesis para comprobar si la variable TimeHoras y la variable Amount son estadísticamente significativas en la materialización de una transacción fraudulenta, es decir relacionar la variable Amount y TimeHoras con la variable Class.
Primero vamos a definir las hipótesis:

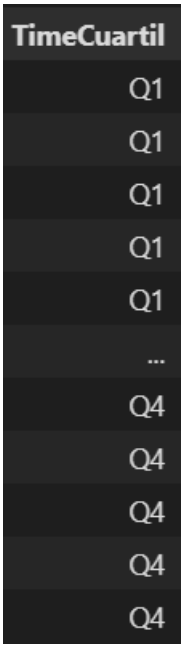
H0: Las variables son independientes (F0 = FE)
HA: Existe una diferencia entre la frecuencia observada y esperada.

- **Variable TimeHoras**

Para poder generar la comparación, al tratarse de una variable cualitativa nominal, y ser un problema de clasificación, es necesario transformar la variable que vamos a relacionar al mismo tipo, se van a generar cuartiles que nos ayudaran a realizar esta comprobación.

```
# Vamos a calcular la probabilidad (odds), la plausibilidad de que un
evento de fraude ocurra a partir del tiempo transcurrido desde
# la primera transaccion, para eso debemos crear una nueva variable a
partir de los cuartiles de la variable Time expresada en hora
(TimeHoras)
# Queremos determinar si la variable TimeHoras tiene relacion con la
materializacion de transacciones de fraude.


transacciones['TimeCuartil'] = pd.qcut(transacciones['TimeHoras'], 4,
labels=['Q1', 'Q2', 'Q3', 'Q4'])
transacciones
```

Figura 19 Codificación de la variables en Cuartiles



Demostración de la creación de la variable TimeCuartil, autoría propia, 2024.

Creamos un gráfico que nos ayude a intuir visualmente si existe una relación entre las variables.

```
# Se genera un grafico de mosaico que nos va a ayudar a determinar
visualmente si existe una relacion entre las dos variables, en este
caso
# la variable Time expresada en horas y la variable Class.


from statsmodels.graphics.mosaicplot import mosaic
plt.figure(figsize=(8,6))
mosaic(transacciones, ['TimeCuartil', 'Class'], title='Gráfico de
Mosaico por Cuartiles de Tiempo y Fraude')
```

Figura 20 Gráfico de mosaico por cuaritles Time Cuartil

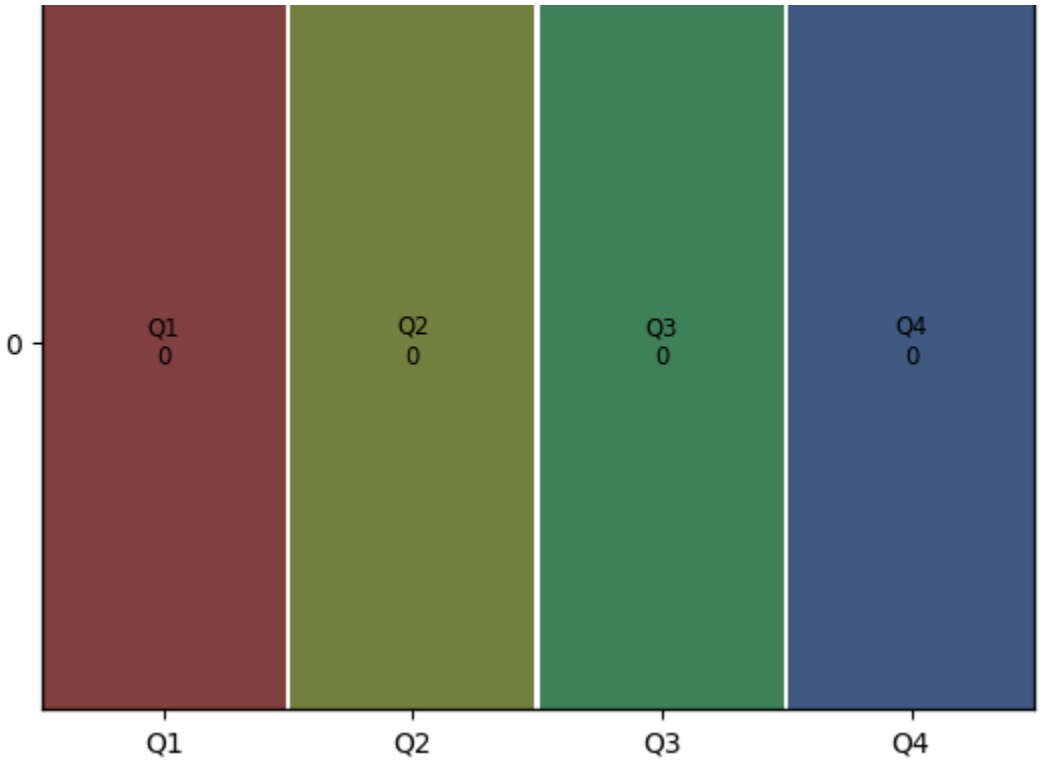


Gráfico de Mosaico TimeCuartil, autoría propia, 2024.

El diagrama de mosaico a relacionado cada uno de los valores de los cuartiles de la variable TimeHoras que es el tiempo expresado en horas con los valores de la variable Class que contiene la clasificación de las transacciones para Fraude o No Fraude, esto lo hace a través de las áreas del gráfico, al verificar el grafico y su salida podemos observar que tienen prácticamente áreas iguales para cada variable y clasificación, debido a que la data está bastante desbalanceada, el gráfico obtenido no nos puede ser de mucha utilidad, se puede observar una ligera diferencia en el tamaño de las áreas de los cuartiles, lo que sugeriría una relación entre la variable TimeHoras y la variable Class, para comprobar si efectivamente existe una relación entre estas dos variables, vamos a realizar una prueba Chi-cuadrado de independencia.

3.3.3 Análisis de Razón de Probabilidades (Odds Ratio)

Antes de calcular el estadístico Chi-Cuadrado primero se quiere comprobar la plausibilidad de ocurrencia de las transacciones de fraude en función de sus clasificación en cuartiles. Generamos una tabla de contingencia y posterior análisis de probabilidades.

```
# Imprimimos la ta0bla de contingencia, con la que posteriormente
haremos el analisis de las probabilidades (Odds)
tabla_contingencia = pd.crosstab(transacciones['TimeCuartil'],
transacciones['Class'])
print(tabla_contingencia)
Class            0    1
TimeCuartil
Q1           71025  177
Q2           71110   92
Q3           71072  129
Q4           71108   94
```

3.3.4 Tablas de Contingencia

Cuartil 1 (Q1)

	OUTCOME +	***OUTCOME -***	***TOTAL***	***RISK***	***ODDS***
Q1 (EXPOSED +)	177	71025	71202	0,0025	0,0025
Q2, Q3, Q4 (EXPOSED -)	315	213290	213605	0,0015	0,0015
TOTAL	492	284315	284807	0,0017	0,0017

Cuartil 2 (Q2)

	OUTCOME +	***OUTCOME -***	***TOTAL***	***RISK***	***ODDS***
Q2 (EXPOSED +)	92	71110	71202	0,0013	0,0013
Q1, Q3, Q4 (EXPOSED -)	400	213205	213605	0,0019	0,0019
TOTAL	492	284315	284807	0,0017	0,0017

Cuartil 3 (Q3)

	OUTCOME +	***OUTCOME -***	***TOTAL***	***RISK***	***ODDS***
Q3 (EXPOSED +)	129	71072	71201	0,0018	0,0018
Q1, Q2, Q4 (EXPOSED -)	363	213243	213606	0,0017	0,0017
TOTAL	492	284315	284807	0,0017	0,0017

Cuartil 4 (Q4)

	OUTCOME +	***OUTCOME -***	***TOTAL***	***RISK***	***ODDS***
Q4 (EXPOSED +)	94	71108	71202	0,0013	0,0013
Q1, Q2, Q3 (EXPOSED -)	398	213207	213605	0,0019	0,0019
TOTAL	492	284315	284807	0,0017	0,0017

Ratios

	RATIO	
CUARTIL	**RISK**	**ODDS**
Q1	1,686	1,687
Q2	0,690	0,690
Q3	1,066	1,066
Q4	0,709	0,708

Los valores obtenidos de Odds Risk de 1,69 indica que las variables que causan el expousure y el outcome están más relacionadas, el valor se encuentra lejos de 1, para el valor de Odds Ratio de 1.69 al ser superior a 1 significa que el evento de transacciones de fraude es más probable que ocurra en el grupo de interés, cuartil 1 y cuartil 3 que en el grupo de comparación. Esto es coherente con el análisis que se realizó respecto a la dispersión de los datos entre el monto y el tiempo y respecto a la distribución que presenta la variable Time. En conclusión, existe una plausibilidad de ocurrencia más probable de transacciones de fraude en los cuartiles 1 y cuartiles 3.

3.3.5 Prueba Chi-Cuadrado

Para generar la prueba se debe importar de la biblioteca scipy la prueba chi2_contingency y se lo realiza con las siguientes líneas de código.

```
# Importar la libreria que contiene la prueba chi cuadrado
from scipy.stats import chi2_contingency

# Para realizar el calculo se utiliza la tabla de contigencia que
creamos anteriormente para el analisis de odds.
chi2, p_val, dof, expected = chi2_contingency(tabla_contingencia) #
pd.crosstab(transacciones['TimeCuartil'], transacciones['Class'])

# Se imprimen los resultados.
print(f"Estadístico de chi-cuadrado: {chi2}")
print(f"Valor p: {p_val}")
print(f"Grados de libertad: {dof}")
print("Frecuencias esperadas:")
print(expected)


Estadístico de chi-cuadrado: 38.71732708858819
Valor p: 1.992248965847955e-08
Grados de libertad: 3
Frecuencias esperadas:
[[71078.99956813   123.00043187]
 [71078.99956813   123.00043187]
 [71078.00129561   122.99870439]
 [71078.99956813   123.00043187]]
```

La hipotesis nula es que las variables son independientes es decir FO = FE mientras que la hipótesis alternativa es que existe una diferencia entre la frecuencia observada y la frecuencia esperada e indicando que las variables se encuentran relacionadas.

H0: Las variables son independientes (F0 = FE)

HA: Existe una diferencia entre la frecuencia observada y esperada

Obtuvimos como estadístico chi-cuadrado el valor de: 38.72, un valor relativamente alto, mientras que para p-value se obtuvo: 0.0000000099224 que es un valor bastante cercano a 0 por lo que se

puede concluir que existe suficiente evidencia estadística para rechazar la hipótesis nula por lo tanto aceptamos la hipótesis alternativa, la variable TimeHoras y Class están relacionadas.

Se debe contrastar el valor de chi-cuadrado obtenido y que este valor no se halle en la cola derecha lo que implicaría que existe una baja probabilidad en la materialización de las frecuencias observadas aun siendo que las variables son independientes. El criterio de la prueba de hipótesis es No descartar HO si el valor calculado < chi_critico.

```
import scipy.stats as stats

# Nivel de confianza
alpha = 0.05
dof = 3

# Calculamos el valor critico
valor_critico = stats.chi2.ppf(1-alpha, dof)
print(f"El valor critico de chi cuadrado es:{valor_critico:.2f}")


El valor critico de chi cuadrado es:7.81
```

El valor obtenido de chi-cuadrado fue de 38.72 que es mayor que el valor critico alpha = 0.05 de 7.815 por lo que se descarta la hipótesis nula de FO = FE y se concluye que las variables están relacionadas, es decir existe una asociación entre la variable *TimeHoras* que esta agrupada por horas y la variable *Class* que indica que una transacción es Fraudulenta o No Fraudulenta.
A la variable TimeCuartil la vamos a codificar para a través del método Label Encoder en Python para que pueda ser utilizada por el algoritmo de clasificación.

```
from sklearn.preprocessing import LabelEncoder
label = LabelEncoder()
transacciones['TimeCuartil'] =
label.fit_transform(transacciones['TimeCuartil'])
```

- **Variable Amount**

De la misma manera vamos a realizar una prueba de independencia para comprobar si la variable

Amount y Class se encuentra relacionadas.

```
# Se genera un gráfico de mosaico que nos va a ayudar a determinar
visualmente si existe una relacion entre las dos variables, en este
caso
# la variable Amount y la variable Class.


from statsmodels.graphics.mosaicplot import mosaic
plt.figure(figsize=(8,6))
mosaic(transacciones, ['AmountCuartil', 'Class'], title='Gráfico de
Mosaico por Cuartiles de Monto y Fraude')
```

Figura 21 Gráfico de mosaico por cuartiles Amount

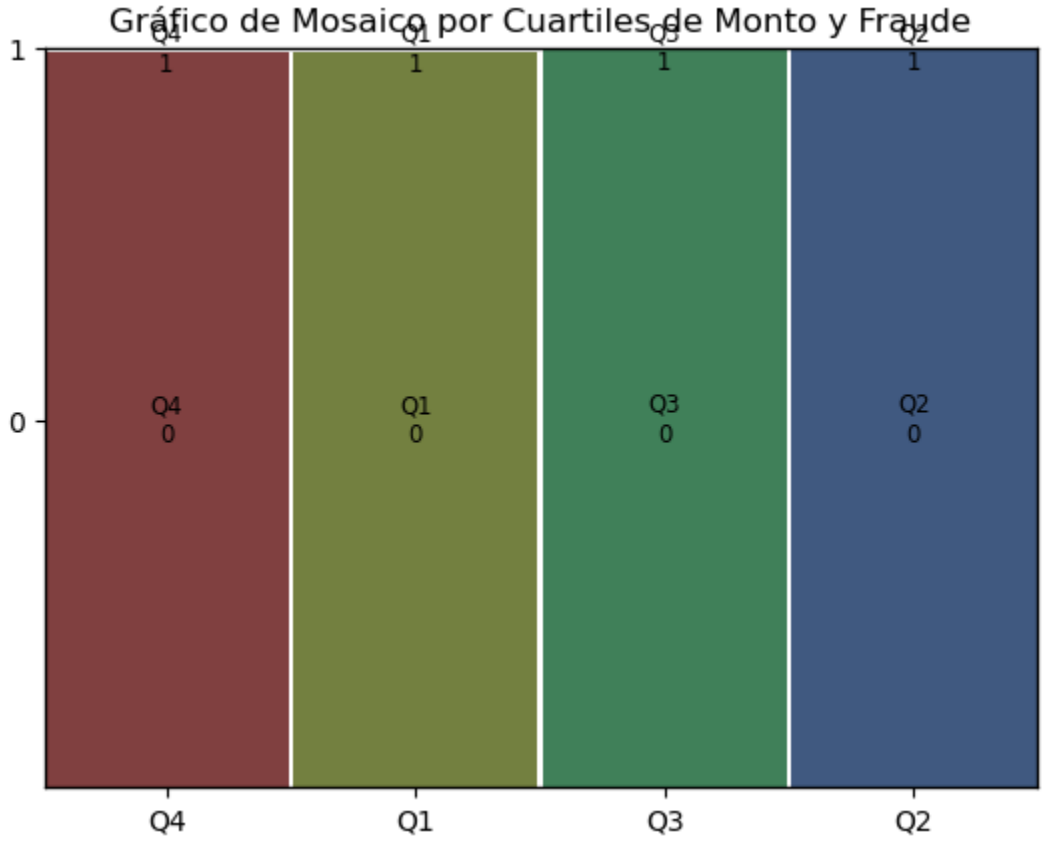


Gráfico de Mosaico AmountCuartil, autoría propia, 2024.

Calculamos el estadístico chi-cuadrado.

```
# Imprimimos la tabla de contingencia, con la que posteriormente
haremos el analisis de las probabilidades (Odds)
tabla_contingencia_2 = pd.crosstab(transacciones['AmountCuartil'],
transacciones['Class'])
print(tabla_contingencia_2)
Class              0    1
AmountCuartil
Q1             71017  224
Q2             71355   47
Q3             70915   47
Q4             71028  174


# Importar la libreria que contiene la prueba chi cuadrado
from scipy.stats import chi2_contingency

# Para realizar el calculo se utiliza la tabla de contigencia que
creamos anteriormente para el analisis de odds.
chi2, p_val, dof, expected = chi2_contingency(tabla_contingencia_2) #
pd.crosstab(transacciones['TimeCuartil'], transacciones['Class'])

# Se imprimen los resultados.
print(f"Estadístico de chi-cuadrado: {chi2}")
print(f"Valor p: {p_val}")
print(f"Grados de libertad: {dof}")
print("Frecuencias esperadas:")
print(expected)

Estadístico de chi-cuadrado: 198.12695588588065
Valor p: 1.0711879000217774e-42
Grados de libertad: 3
Frecuencias esperadas:
[[71117.93219619   123.06780381]
 [71278.654071     123.345929  ]
 [70839.41416468   122.58583532]
 [71078.99956813   123.00043187]]
```

Los estadísticos obtenidos, chi cuadrado de 198,12 y el valor de P con un valor muy cercano a 0 nos indican que existe suficiente evidencia estadística para rechazar la hipótesis nula de FO = FE por lo que aceptamos la hipótesis alternativa y concluimos que existe una relación entre el monto y la materialización de una transacción como fraudulenta o no fraudulenta.

```
import scipy.stats as stats

# Nivel de confianza
alpha = 0.05
dof = 3

# Calculamos el valor critico
valor_critico = stats.chi2.ppf(1-alpha, dof)
print(f"El valor critico de chi cuadrado es:{valor_critico:.2f}")
El valor critico de chi cuadrado es:7.81
```

De la misma manera al contrastar el valor de chi cuadrado obtenido contra el valor de chi cuadrado critico podemos concluir que efectivamente existe una relación entre las variables *Amount* y *Class*.

Adicionalmente verificamos la presencia de valores atípico a través de del diagrama de cajas creado, el diagrama también nos da una idea de que la distribución de los datos podría ser no normal.

Para este caso vamos primero vamos a generar una transformación logarítmica base 10 para ajustar un poco la distribución de la variable, se consideró que existen valores 0 en la columna por lo que se creó una función para que retorne el valor 0 y que no aplique la transformación sobre esas observaciones ya que si aplicamos una transformación logarítmica a valores 0 el valor resultante es infinito para después realizar el proceso de normalización de la variable.

```
# Funcion para la transformación logarítmica de la variable Amount
def safe_log10(x):
    if x > 0:
        return np.log10(x)
    else:
        return 0  # Retorna 0 para valores no válidos

transacciones['Amount'] = transacciones['Amount'].apply(safe_log10)
```

3.3.6 t-SNE

Para las variables V1, V2, V3, … V28 que son el resultado de la aplicación de una técnica de Reducción de dimensiones PCA vamos a utilizar una técnica de visualización de datos de alta complejidad, esta técnica se denomida t-SNE (t-distributed Stochastic Neighbor Embedding) que

es una técnica de aprendizaje no supervisado de reducción no lineal de alta dimensionalidad y que suele utilizarse para visualizar conjuntos de datos complejos en pocas dimensiones (Awan, 2024), esto nos va a permitir comprender mejor los patrones y las relaciones en los datos, e identificar si efectivamente existen relaciones complejas, estas relaciones serian difíciles de capturar para los modelos de clasificación más tradicionales.

Creamos el nuevo dataset que solo contenga las variables V's y la clase.

```
x_tsne =
transacciones.drop(['Time','Amount','Class','TimeHoras','AmountCuartil
','TimeCuartil'], axis=1)
y_tsne = transacciones['Class']
```

Importamos la librerías con las que vamos a trabajar, también generamos la estandarización de los datos a través del método scaler.fit_trasform(), dejando los datos con media igual a cero y con desviación estándar de uno.

```
from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler

# Normalizar los datos
scaler = StandardScaler()
X_scaled = scaler.fit_transform(x_tsne)
```

```
X_scaled
array([[-0.69424232, -0.04407492,  1.6727735 , ..., -0.39217043,
         0.33089162, -0.06378115],
       [ 0.60849633,  0.16117592,  0.1097971 , ...,  0.26106948,
        -0.02225568,  0.04460752],
       [-0.69350046, -0.81157783,  1.16946849, ..., -0.28844675,
        -0.13713686, -0.18102083],
       ...,
       [ 0.98002374, -0.18243372, -2.14320514, ..., -0.18118178,
         0.01103672, -0.0804672 ],
       [-0.12275539,  0.32125034,  0.46332013, ...,  1.133635  ,
         0.26960398,  0.31668678],
       [-0.27233093, -0.11489898,  0.46386564, ..., -1.69685342,
        -0.00598394,  0.04134999]])
```

Implementamos el algoritmo.

```
# Implementar t-SNE
tsne = TSNE(n_components=2, perplexity=30, learning_rate=200,
n_iter=1000, random_state=123)
X_tsne = tsne.fit_transform(X_scaled)
```

Creamos un dataframe con el resultado obtenido de la aplicación del algoritmo de t-SNE, esto para poder generar un gráfico para interpretar.

```
# Crear un DataFrame con los resultados de t-SNE
tsne_df = pd.DataFrame(X_tsne, columns=['TSNE1', 'TSNE2'])
tsne_df['Class'] = y_tsne
```

Generamos el gráfico.

```
# Visualizar los resultados
plt.figure(figsize=(10, 8))

sns.scatterplot(x='TSNE1', y='TSNE2', hue='Class', palette='deep',
data=tsne_df[tsne_df['Class'] == 0],
                s=25, alpha=0.5)

# Graficar los puntos de la clase 1 (fraude) con mayor tamaño y menor
transparencia
sns.scatterplot(x='TSNE1', y='TSNE2', hue='Class',
palette='Oranges',data=tsne_df[tsne_df['Class'] == 1],
                s=25, alpha=1)
plt.title('Visualización de t-SNE')
plt.show()
```

Figura 22 Visualización del resultado del algoritmo t-SNE

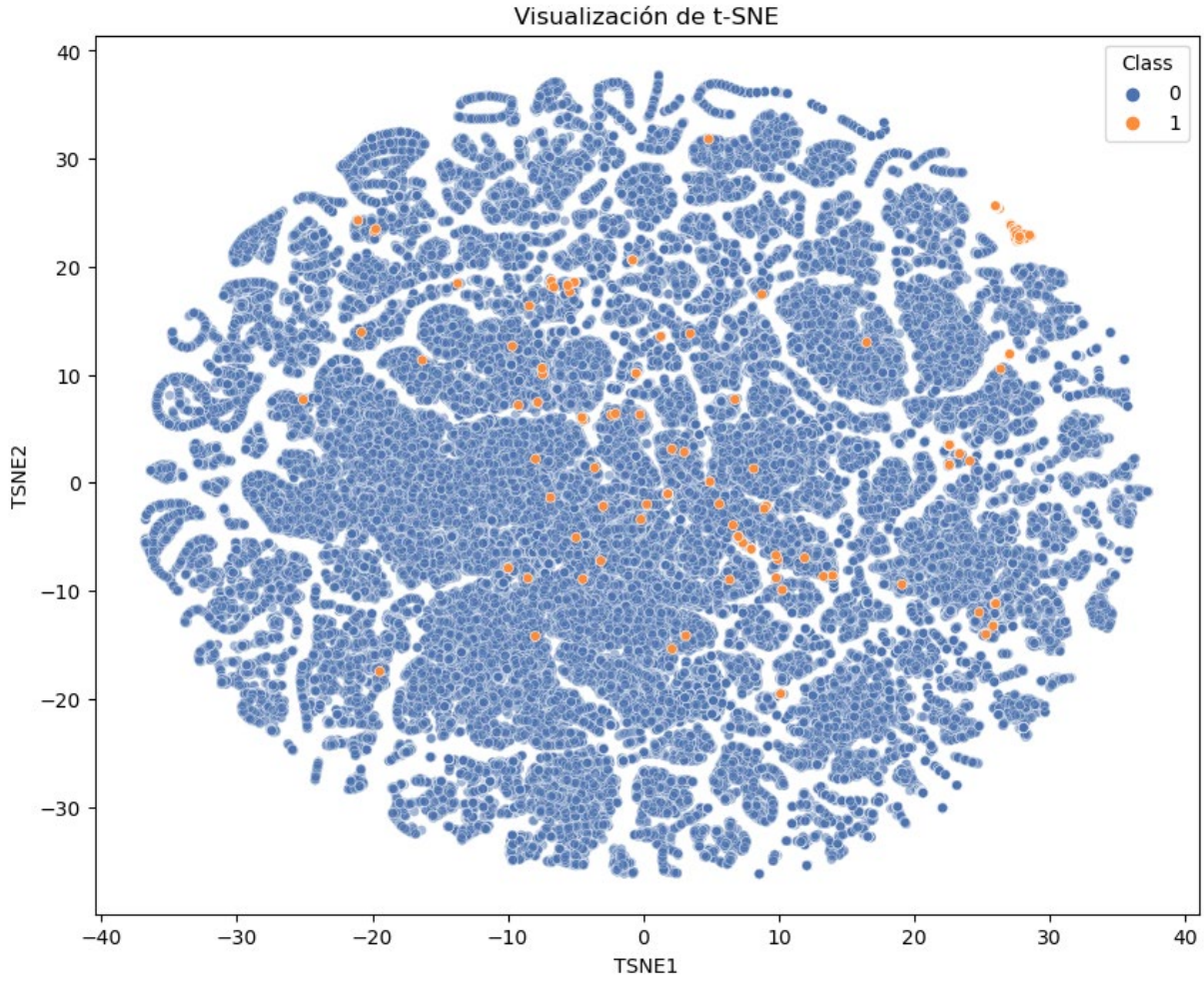


Gráfico de t-SNE, autoría propia, 2024.

En la visualización obtenida se puede observar cómo existe una gran densidad de puntos azules que representan las transacciones normales y como a través de estos se distribuyen las transacciones de fraude en color naranja, las transacciones de fraude tienen dos comportamientos, en algunas zonas se agrupan mientras que en otras zonas se van distribuyendo, están dispersas habiendo lugares en donde no se distinguen consumos fraudulentos, también se puede identificar que las transacciones de fraude a menudo se encuentran cerca de las transacciones legitimas, identificando que las transacciones de fraude imitan comportamiento de las transacciones normales, al observar la dispersión de las transacciones de fraude y como están dispersas en ocasiones en grupos, podemos establecer que existen patrones complejos en la data y que para

identificar estos patrones complejos es útil usar un algoritmo de redes neuronales que capture de mejor manera estos patrones.

3.4 Normalización de los datos

Debido a la escala de la variable Amount y por motivos de que el modelo tenga un mejor rendimiento vamos a realizar una normalización de los datos de la variable. Vamos a utilizar la normalización Min-Max para transformar los valores a una escala de valores entre 0 y 1 y para que permanezcan en una escala similar al resto de variables.

$$x_{norm} = \frac{x_i - x_{mix}}{x_{max} - x_{mix}}$$

- **Normalización variable Amount**

Calculamos y definimos las variables min y max.

```
min_a = transacciones['Amount'].min()
max_a = transacciones['Amount'].max()
print(f"El valor minimo de la columna Amount es:{min_a}")
print(f"El valor maximo de la columna Amount es:{max_a:.2f}")
El valor minimo de la columna Amount es:-2.0
El valor maximo de la columna Amount es:4.41

# Realizamos el calculo
transacciones['Amount']= (transacciones['Amount']-min_a)/(max_a-min_a)
```

Las variable Amount se ha normalizada en un a escala entre 0 y 1

Figura 23 Normalización de la variable Amount

Amount
0.651346
0.379069
0.714260
0.638347
0.599870
...
0.294314
0.529546
0.597796
0.468035
0.676538

Resultado de la normalización Min-Max, autoría propia, 2024

3.5 Creación del dataset para el entrenamiento

Una vez trabajadas todas la variables a través del preprocesamiento, establecemos los características con la que vamos a generar el set de datos para entrenar el modelo, descartamos las variables Time, TimeHoras y AmountCuartil, que fueron creadas para realizar los cálculos estadísticos.

```
# Eliminamos las variables con el metodo drop
auto_transacciones = transacciones.drop(['Time', 'TimeHoras',
'AmountCuartil'], axis=1)
```

Creamos el archivo .csv con el vamos a entrenar el modelo.

```
# Guardamos el archivo en formtato csv
nombre_archivo = 'auto_transacciones.csv'
auto_transacciones.to_csv(nombre_archivo, index=False)
```

Figura 24 Dataset listo para el entrenamiento

...	V22	V23	V24	V25	V26	V27	V28	Amount	Class	TimeCuartil
...	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	0.651346	0	0
...	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	0.379069	0	0
...	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	0.714260	0	0
...	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	0.638347	0	0
...	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	0.599870	0	0
...	...	...	...	...	...	...	...	...	...	...
...	0.111864	1.014480	-0.509348	1.436807	0.250034	0.943651	0.823731	0.294314	0	3
...	0.924384	0.012463	-1.016226	-0.606624	-0.395255	0.068472	-0.053527	0.529546	0	3
...	0.578229	-0.037501	0.640134	0.265745	-0.087371	0.004455	-0.026561	0.597796	0	3
...	0.800049	-0.163298	0.123205	-0.569159	0.546668	0.108821	0.104533	0.468035	0	3
...	0.643078	0.376777	0.008797	-0.473649	-0.818267	-0.002415	0.013649	0.676538	0	3

Muestra del dataset creado para el entrenamiento, autoría propia, 2024

CAPITULO IV. ENTRENAMIENTO DEL MODELO.

4.1 División del conjunto de datos

Para el entrenamiento del modelo vamos a dividir el conjunto de datos en dos partes en la siguiente proporción: 80% para el entrenamiento y 20% para el test, debido a que el objetivo del autoencoder es detectar anomalías para identificar transacciones de fraude, únicamente vamos a trabajar con la clase mayoritaria, transacciones no fraudulentas = 0, dado que se pretende trabajar en función de la reconstrucción del error para detectar transacciones que presentan anomalías.

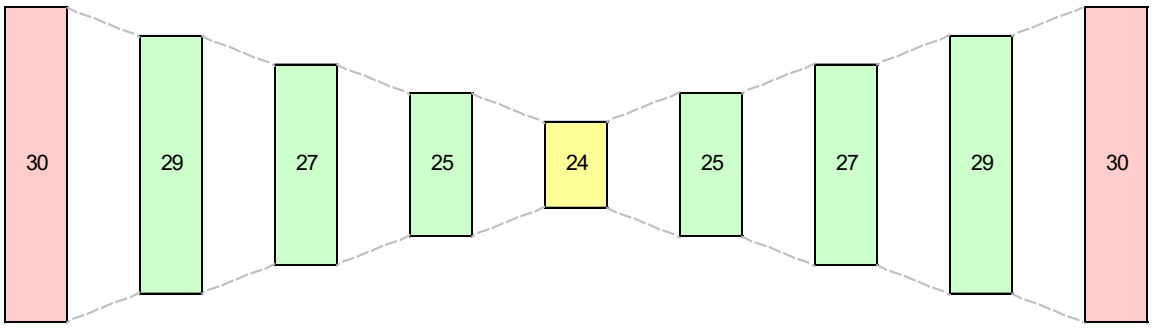
```
from sklearn.model_selection import train_test_split
X_train, X_test = train_test_split(auto_transacciones, test_size=0.2,
random_state=123)
# Solo clases normales para medir el error de reconstruccion

X_train = X_train[X_train.Class == 0]
X_train = X_train.drop(['Class'], axis=1)
X_train = X_train.values
X_train

Y_test = X_test['Class']
X_test = X_test.drop(['Class'], axis=1)
X_test = X_test.values
```

La idea principal es reconstruir transacciones a partir de los datos de entrenamiento e identificar las anomalías cuando el error de reconstrucción sea demasiado alto, esto a través de definir un umbral.

4.2 Diseño de la arquitectura del Autoencoder

La arquitectura del autoencoder va a estar compuesta de una capa de entrada con 30 características, una para cada variable del set de datos, 7 capas ocultas, y una capa de salida con 30 características reconstruidas.

Figura 25 Arquitectura del Autoencoder



Arquitectura del Autoencoder, autoría propia, 2024

Implementamos el modelo.

```
import numpy as np
from keras.layers import Input

np.random.seed(123)
dimension = X_train.shape[1]
capa_entrada = Input(shape=(dimension,))
capa_entrada
import numpy as np
from keras.layers import Input

np.random.seed(123)
dimension = X_train.shape[1]
capa_entrada = Input(shape=(dimension,))
capa_entrada


<KerasTensor shape=(None, 30), dtype=float32, sparse=None,
name=keras_tensor_319>
```

Arquitectura de los autoencoders en Python

```
from keras.models import Sequential
from keras.layers import Dense
from tensorflow.keras.regularizers import l2
from keras.layers import Dense
encoder = Dense(29, activation =
'tanh',kernel_regularizer=l2(l2=0.01))(capa_entrada)
encoder = Dense(27, activation = 'tanh')(encoder)
encoder = Dense(25, activation = 'tanh')(encoder)
```

```
encoder = Dense(24, activation = 'relu')(encoder)


# Decoder

decoder = Dense(25, activation = 'relu')(encoder)
decoder = Dense(27, activation = 'relu')(decoder)
decoder = Dense(29, activation = 'relu')(decoder)
output_layer = Dense(30, activation='tanh')(decoder)
```

4.3 Ajuste de hiperparametros

4.3.1 Regularización.

Se utilizo regularización L2 Ridge para evitar el sobre ajuste del modelo al añadir un término Lambda que pretende reducir de forma proporcional el valor de todos los coeficientes del modelo. El grado de penalización es controlado por el hiperparámetro Lambda. (Rodrigo, 2016)

4 3.2 Optimizador.

Se utilizó el optimizador ADAM o Adaptative Moment Estimation que es un algoritmo de optimización utilizado en redes neuronales que permite la actualización de los parámetros del modelo en cada iteración. Adam adapta la tasa de aprendizaje de cada parámetro individualmente en función de su estimación del momento y de la magnitud del gradiente, esto permite que el modelo se ajuste de manera más eficiente y efectiva a los datos de entrenamiento. (Interactive Chaos, s.f)

4.3.3 Learning Rate.

Se utilizó una tasa de aprendizaje de 0.01

4.3.4 Función de activación.

Se utilizaron funciones de activación Tahn y ReLu para las salidas de cada una de las capas ocultas del autoencoder.

4.3.5 Función de Perdida.

Debido a que vamos a trabajar con la reconstrucción en el error, vamos a trabajar con la función de perdida MSE que pretende disminuir el cuadrado de las diferencias entre los valores reales y los predichos.

4.3.4 Early Stopping

El early stopping es una técnica implementada en la biblioteca de Keras que permite detener automáticamente el entrenamiento de un modelo si este no mejora.

4.3.5 Tamaño de lote

Se definió el tamaño del lote para cada iteración de 4096

4.3.6 Número de Épocas.

Se estableció un número de épocas máximo de 1000, hay que tomar en cuenta que se está trabajando con early stopping.

```
# Creacion del modelo
from keras.models import Model
autoencoder = Model(inputs = capa_entrada, outputs = output_layer)


# Optimizador
from keras.optimizers import Adam
autoencoder.compile(optimizer = Adam(learning_rate = 0.01), loss =
'mse')


# Earlly Stopping
from tensorflow.keras.callbacks import EarlyStopping,
ReduceLROnPlateau
early_stopping = EarlyStopping(
    monitor='val_loss',       # Monitoriza la pérdida de validación
    patience=20,              # Número de epochs sin mejora antes de
detener el entrenamiento
    verbose=1,                # Mostrar información sobre el paro
anticipado
    restore_best_weights=True,# Restaurar los pesos del modelo que
dieron el mejor rendimiento en validación
)
```

4.4 Proceso de entrenamiento

```
nits = 1000
tam_lote = 4096 #1024
history = autoencoder.fit(X_train, X_train, epochs = nits, batch_size
= tam_lote, # en la epoca 4 ya esta, para mse
                shuffle = True, validation_data=(X_test, X_test),
verbose = 1,
                callbacks=[early_stopping, tensorboard_callback])
```

Figura 26 Proceso de entrenamiento del Autoencoder

Proceso de entrenamiento del Autoencoder, autoría propia, 2024

Realizamos predicciones con el modelo entrenado.

```
# Prediciones
X_pred = autoencoder.predict(X_test)
X_pred
```

Cálculo del MSE y establecimiento del umbral para la reconstrucción del error.

```
# MSE
ecm = np.mean(np.power(X_test - X_pred, 2), axis=1)
ecm


# Umbral
umbral_fijo = 0.85
Y_pred = [1 if e > umbral_fijo else 0 for e in ecm]
```

CAPITULO V. EVALUACIÓN DE LA EFICIENCIA DEL MODELO

5.1 Métricas especificas

Dado que nos interesa generar un modelo que identifique y clasifique correctamente transacciones generadas con tarjeta de crédito y lo que nos interesa es que clasifique correctamente las instancias positivas, es decir las transacciones fraudulentas y que al mismo tiempo detecte la mayor cantidad de verdaderos negativos para reducir la cantidad de falsos negativos las métricas que vamos a utilizar son la Exactitud, la Sensibilidad y la Especificidad. Debido al alto desbalance que presentan los datos, la métrica del F1-Score no resulta útil para este caso.

Para una institución financiera es más importante detectar transacciones fraudulentas y al mismo tiempo disminuir el número de transacciones que son clasificadas erróneamente como transacciones validas siendo estas transacciones fraudulentas, en términos económicos, la materialización de transacciones fraudulentas tiene un costo más elevado que el dinero que deja de percibir por clasificar una transacción de manera incorrecta como falso positivo.

5.1.2 Matiz de Confusión

	0 (-)	**1 (+)**
0 (-)	53951	2896
1 (+)	12	103

5.1.3 Exactitud.

$$\frac{TP + TN}{TP + TN + FP + FN}$$

La exactitud del modelo es de 0.95 esto indica que el modelo es capaz de clasificar correctamente el 95% de las veces, esto es congruente con los valores de falsos positivos obtenidos que son 2896 instancias y que representan alrededor del 5% de los verdaderos positivos (53951). Se debe considerar que el dataset está altamente desbalanceado lo que puede influir en que el modelo tenga este número tan alto de falsos positivos.

5.1.3 Sensibilidad

$$\frac{TP}{TP + FN}$$

La sensibilidad obtenida para la clase positiva fue de 0.90, la sensibilidad nos entrega la cobertura de la muestra positiva real, es decir que tanto en porcentaje está clasificando correctamente como una instancia fraudulenta, el modelo clasifica las instancias positivas el 90% de las veces al mismo tiempo que mantiene pocas instancias como falsos negativos, que en el contexto del problema es un buen indicador debido que poquísimas transacciones las que han sido clasificadas como no fraudulentas siendo que son de fraude.

5.1.4 Especificidad

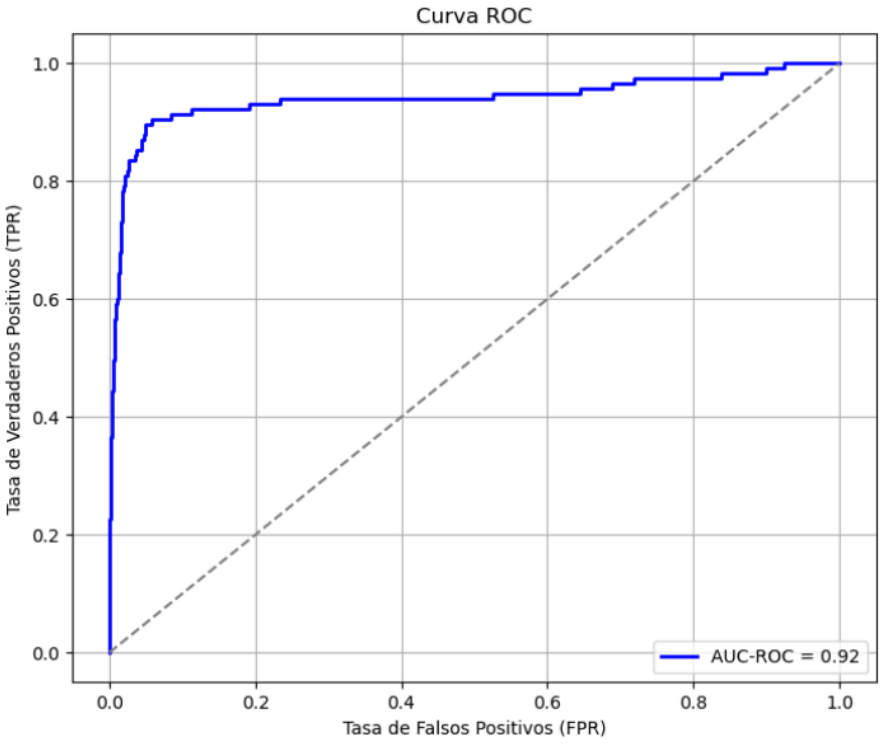
$$\frac{TN}{TN + FP}$$

Por último, calculamos la especificad para saber que tanto nuestro modelo clasifica correctamente las instancias verdaderas negativas, es decir los no fraudes. Se obtuvo una métrica del 0.95 lo que indica que el modelo tiene una alta capacidad para identificar correctamente las muestras negativas.

5.2 Curva ROC

La curva es un gráfico que nos permite graficar la tasa de verdaderos positivos TPR y la tasa de falsos positivos FRP y nos muestra el rendimiento del modelo en todos los umbrales de clasificación.

Figura 27 Curva ROC



Curva ROC del modelo, autoría propia, 2024

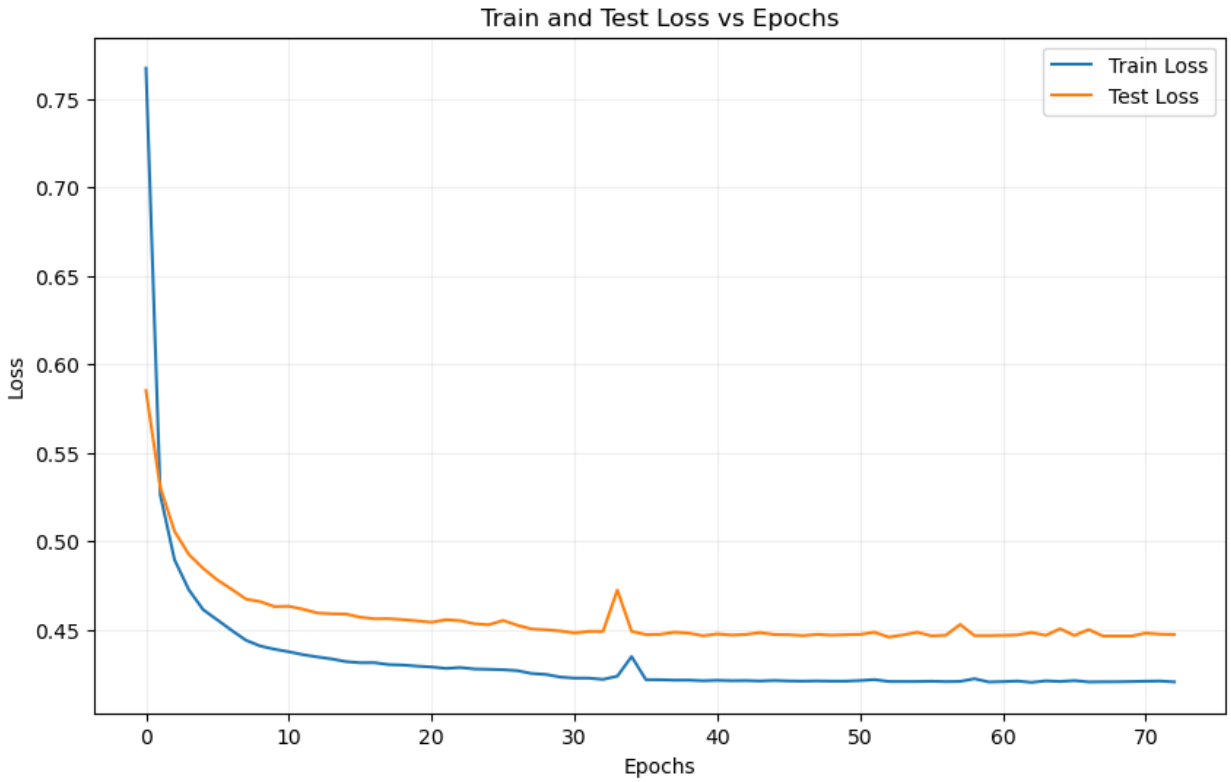
Podemos observar que la métrica AUC-ROC se acerca bastante a 1 (0.92) lo que significa que el existe una buena separabilidad entre las clases esto implica que es capaz de distinguir entre la clase negativa y la positiva.

```
# Calcular las curvas ROC y Precision-Recall
from sklearn.metrics import roc_curve
from sklearn.metrics import precision_recall_curve, roc_auc_score
```

```
fpr, tpr, roc_thresholds = roc_curve(Y_test, ecm)
precision_vals, recall_vals, pr_thresholds =
precision_recall_curve(Y_test, ecm)

# Calcular el AUC-ROC
auc_roc = roc_auc_score(Y_test, Y_pred)

# Graficar la curva ROC
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.plot(fpr, tpr, color='blue', lw=2, label=f'AUC-ROC =
{auc_roc:.2f}')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.xlabel('Tasa de Falsos Positivos (FPR)')
plt.ylabel('Tasa de Verdaderos Positivos (TPR)')
plt.title('Curva ROC')
plt.legend(loc='lower right')
plt.grid(True)

# Graficar la curva Precision-Recall
plt.subplot(1, 2, 2)
plt.plot(recall_vals, precision_vals, color='blue', lw=2)
plt.xlabel('Recuperación (Recall)')
plt.ylabel('Precisión (Precision)')
plt.title('Curva Precision-Recall')
plt.grid(True)

# Mostrar gráficos
plt.tight_layout()
plt.show()
```

5.3 Entrenamiento del Modelo

Figura 28 Error y Error de validación en el entrenamiento



Tran and Test Loss vs Epochs, autoria propia, 2024

Se puede observar que existe una disminución continua de la perdida hasta llegar a un punto donde se ha estabilizado, esto sugiere que el modelo ha aprendido y se ha ajustado bien a los parámetros. Se evidencia que existe convergencia del modelo tras un número especifico de épocas, al aplicar la técnica del early stopping, el modelo ajusto los pesos en la época que tuvo mejor desempeño que fue la época 53. El grafico de entrenamiento no presenta oscilaciones significativas lo que indica que el aprendizaje fue estable durante todo el entrenamiento. Tampoco existe evidencia de que haya un sobreajuste en el modelo.

5.4 Presentación de los resultados.

```
from sklearn.metrics import classification_report
print(classification_report(Y_test, Y_pred))
  precision    recall  f1-score   support

           0       1.00      0.95      0.97     56847
           1       0.03      0.90      0.07       115

    accuracy                           0.95     56962
   macro avg       0.52      0.92      0.52     56962
weighted avg       1.00      0.95      0.97     56962
```

5.5 Identificación de Limitaciones y Propuestas de Mejora.

Durante el proceso de desarrollo de este proyecto se identificaron varias limitaciones respecto al desarrollo correcto y la implementación del modelo con el lenguaje de programación utilizado.

A continuación, se exponen algunas de la limitaciones encontradas y su respectiva propuesta de mejor.

Lenguaje de programación utilizado.

Python es un lenguaje de programación de alto nivel muy completo y versátil para realizar varias tareas en distintos ámbitos, para el caso de la ciencia de datos, resulta ser un lenguaje muy útil, sin embargo, durante la elaboración del proyecto me encontré en algunas situaciones en donde la falta de conocimientos de la herramienta me limitó en los objetivos que tenía, por ejemplo en el ámbito del análisis estadístico existen otras herramientas mejor optimizadas para estas tareas, la propuesta para futuros proyectos es trabajar el análisis estadístico en lenguaje R y a partir del dataset final obtenido generar los modelos de Deep Learning en Python.

Limitaciones respecto a los recursos.

Una limitación importante para el desarrollo del proyecto fue encontrar los datos precisos para generar un buen algoritmo de clasificación, el dataset utilizado puede servir para explorar y realizar pruebas de cómo se debería implementar un determinado algoritmo para un problema de clasificación, sin embargo en el mundo real existen muchas más características y dimensiones para las cuales el científico de datos debe estar preparado, una propuesta de mejora es desarrollar el mismo tipo de algoritmo con otros datasets.

También se puede complementar el modelo desarrollado con algoritmos de clasificación más tradicionales como la regresión logística o los árboles de decisión, se puede hacer una combinación de algoritmos con el objetivo de mejorar las métricas y por mejorar la clasificación que se está realizado. Explorar nuevas alternativas de algoritmos y combinaciones que mejoren la clasificación.

Limitaciones respecto al conocimiento

El proyecto desarrollado requiere de un alto conocimiento en técnicas de aprendizaje profundo, generar un buen algoritmo de clasificación basado en redes neuronales demanda un amplio conocimiento en varios temas por parte del científico de datos, es indispensable nunca dejar de auto educarse, tener curiosidad científica e indagar sobre los temas que nos interesan, solo de esta manera se pueden mejorar modelos como el que aquí se ha presentado y mejorar como profesional de la ciencia de los datos.

CAPITULO VI. CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES

El modelo de detección de fraudes basado en autoencoders demostró una alta eficiencia en la identificación de transacciones fraudulentas apestar del desbalance que existen en los datos, se implementaron técnicas de regularización que mejoraron significativamente la capacidad de generalización del conocimiento.

El desbalance significativo en el conjunto de datos, una proporción de muestras no fraudulentas mucho mayor que las fraudulentas, ha afectado la precisión global del modelo, reflejando la necesidad de técnicas avanzadas para mitigar el impacto de este desbalance. El ajuste de los umbrales y la implementación de autoencoders mejoró la detección, pero mitigar la cantidad de falsos positivos todavía sigue siendo un reto.

La implementación del modelo de detección de fraude a un sistema de producción real podría ser factible, dadas las métricas obtenidas como el recall y la especificidad, el problema de los falsos positivos puede ser trabajado bajo la supervisión de un grupo de analistas. Para la implementación del modelo sería conveniente alinearse a la metodología escogida MLOps ya que permite un monitoreo del algoritmo y una mejora e integración continua sobre el modelo.

El análisis t-SNE (t-distributed Stochastic Neighbor Embedding) realizado permitió la visualización de la estructura intrínseca de los datos de alta dimensión proyectarlos en un espacio de menor dimensión. A través de esta visualización se reveló que las transacciones fraudulentas y no fraudulentas están dispersas por todas las variables existiendo también superposición, esto justificó la necesidad de utilizar un modelo capaz de capturar relaciones no lineales complejas, como los autoencoders. También se proporcionaron análisis visuales para sustentar la complejidad de los datos.

El análisis chi-cuadrado realizado en las variables relacionadas con las transacciones confirmó la existencia de una relación de dependencia significativa entre ciertas características de las transacciones y la materialización de transacciones de fraude. Esto sugiere que las variables

seleccionadas en el modelo son relevantes para la detección de patrones de fraude, validando así su inclusión en el modelo de autoencoders.

5.2 RECOMENDACIONES

El desbalanceo en la data ocasionó que algunas métricas de evaluación no puedan ser consideradas para medir correctamente la efectividad del modelo, para trabajar datasets desbalanceados se recomienda aplicar técnicas de sobre muestreo o submuestreo para igualar las clases ya sea eliminando instancias de la clase mayoritaria o duplicando instancias de la clase minoritaria, se recomienda probar con la técnica como el SMOTE (Synthetic Minority Over-sampling Technique) que crea nuevas muestras sintéticas a partir de las existentes con el fin de igualar la clases, esto con el fin de mejorar las métricas afectadas por el desbalance y mejorar la calidad del modelo.

Se recomienda considerar un enfoque de algoritmos combinados, es decir combinar los autoencoders con otro tipo de algoritmos de clasificación como la regresión logística o los árboles de decisión para mejorar la precisión del modelo y minimizar los falsos positivos.

Se recomienda utilizar el enfoque de MLOps para asegurar que el modelo se despliegue y se mantenga de manera eficiente y escalable en un entorno de producción. Se recomienda implementar una automatización a través de pipeplines desde el procesamiento de datos y el reentrenamiento del modelo, garantizando un flujo de trabajo con el mínimo número de errores desde la ingesta de los datos hasta la tarea de clasificación del algoritmo.

Bibliografía y referencias.

1. Asociación de Bancos del Ecuador. (2024, Marzo 18). *Boletín Macroeconómico Marzo 2024*. https://asobanca.org.ec/wp-content/uploads/2024/03/Boletin-macroeconomico-Marzo-2024.pdf

2. Asociación de Bancos del Ecuador. (2023, Septiembre 18). Consumos realizados con tarjetas de crédito llegan a USD 10.382 millones en el primer semestre de 2023. https://asobanca.org.ec/wp-content/uploads/2023/09/2023-09-18-BP-ABC-de-las-Tarjetas-Asobanca-Aval-Buro.pdf

3. Pérdidas por fraude con tarjeta superarán los $ 32 mil millones en 2021. (2021, Abril 19). Revista Gestión. https://revistagestion.ec/cifras/perdidas-por-fraude-con-tarjeta-superaran-los-32-mil-millones-en-2021/

4. Baesens, B., Vlasselaer, V. V., & Verbeke, W. (2015). *Fraud analytics using descriptive, predictive, and social network techniques: A guide to data science for fraud detection*. John Wiley & Sons.

5. Awoyemi, J. O., Adetunmbi, A. O., & Oluwadare, S. A. (2017). Credit card fraud detection using machine learning techniques: A comparative analysis. *2017 International Conference on Computing Networking and Informatics (ICCNI)*. https://doi.org/10.1109/iccni.2017.8123782

6. Dornadula, V. N., & Geetha, S. (2019). Credit card fraud detection using machine learning algorithms. *Procedia Computer Science*, *165*, 631-641. https://doi.org/10.1016/j.procs.2020.01.057

7. Asale, R.-., & Rae. (s. f.). *fraude | Diccionario de la lengua española*. «Diccionario de la Lengua Española» - Edición del Tricentenario. https://dle.rae.es/fraude

8. Mantuano, G. (2021, May 8). *Una red está a la caza de las tarjetas de crédito*. El Diario Ecuador. https://www.eldiario.ec/actualidad/manabi/una-red-esta-a-la-caza-de-las-tarjetas-de-credito/

9. *¿Qué es el machine learning (ML)? | IBM. (s. f.). https://www.ibm.com/es-es/topics/machine-learning*

10. *DataScientest.com. (2023, 30 octubre). Machine Learning: definición, funcionamiento, usos. Formación En Ciencia de Datos | DataScientest.com. https://datascientest.com/es/machine-learningdefinicionfuncionamientousos#:~:text=El%20Machine%20Learning%20o%20aprendizaje,%2C%20im%C3%A1genes%2C%20estad%C3%ADsticas%2C%20etc*

11. Metaphorce. (2022, 18 agosto). *Descubre todo lo que necesitas saber sobre el Machine Learning*. https://www.linkedin.com/pulse/descubre-todo-lo-que-necesitas-saber-sobre-el-machine-learning-/

12. Gonzalez, F. (s. f.). *Machine Learning: La base que tenes que tener*. https://somospnt.com/blog/53-introduccion-a-machine-learning

13. *Gonzalez, J. L. (2020, 13 julio). Tipos de aprendizaje automático - SoldAI - Medium. Medium. https://medium.com/soldai/tipos-de-aprendizaje-autom%C3%A1tico-6413e3c615e2*

14. *Deep learning. (s. f.). MATLAB & Simulink. https://la.mathworks.com/discovery/deep-learning.html*

15. *BBVA. (2023, 20 noviembre). ¿Qué es el 'deep learning' y cómo beneficia nuestro día a día? BBVA NOTICIAS. https://www.bbva.com/es/innovacion/que-es-el-deep-learning-y-como-beneficia-nuestro-dia-a-dia/*

16. *Estructura de una red neuronal | Interactive Chaos. (s. f.).* https://interactivechaos.com/es/manual/tutorial-de-machine-learning/estructura-de-una-red-neuronal

17. *Xeridia. (2023, 15 junio). Redes Neuronales artificiales: Qué son y cómo se entrenan.* https://www.xeridia.com/blog/redes-neuronales-artificiales-que-son-y-como-se-entrenan-parte-i

18. *James, G., Witten, D., Hastie, T., Tibshirani, R., & Taylor, J. (2023). An introduction to statistical learning: With applications in Python. Springer.*

19. *Unir, V. (2023, 14 marzo). ¿Qué es el algoritmo backpropagation para el entrenamiento de redes neuronales? UNIR.* https://www.unir.net/ingenieria/revista/backpropagation/

20. *Vercaca. (s. f.). GitHub - Vercaca/NN-Backpropagation: Implement a Neural Network trained with back propagation in Python. GitHub. https://github.com/Vercaca/NN-Backpropagation*

21. *Ahmad, M. N., Mahmood, A. K., Hashim, K. F., Mustakim, F. B., Selamat, A., Bajuri, M. Y., & Arshad, N. I. (2021). Artificial intelligence model and correlation for characterization and viscosity measurements of mono & hybrid nanofluids concerned graphene oxide/silica. Journal of Thermal Analysis and Calorimetry, 145(4), 2209-2224.* https://doi.org/10.1007/s10973-021-10687-5

22. *Cañadas, R. (2021, August 30). Autoencoders | Que son, Arquitectura Y sus Aplicaciones. abdatum. https://abdatum.com/tecnologia/autoencoders*

23. Sharma, N. S. (2018, November 28). *Deep Learning Architectures for Payments Fraud Detection. AI & Big Data Expo North America 2018.* https://www.ai-expo.net/northamerica/wp-content/uploads/2018/11/1700-Nitin-Sharma-Paypal-AI-ENT-V1.pdf

24. Medeiros, I. (2023, July 28). *MLOps spanning whole machine learning life cycle: Paper summary*. MarkTechPost. https://www.marktechpost.com/2023/07/28/mlops-spanning-whole-machine-learning-life-cycle-paper-summary/

25. *El concepto de MLOps*. (2023, Septiembre 23). Red Hat - We make open source technologies for the enterprise. https://www.redhat.com/es/topics/ai/what-is-mlops

26. *¿Qué es DevOps? | IBM*. (s. f.). https://www.ibm.com/mx-es/topics/devops

27. *ILIMIT. (2020, 10 Diciembre). Integración continua, entrega continua y despliegue continuo https://ilimit.com/blog/integracion-continua-entrega-continua-despliegue-continuo/*

28. *Colaboradores de Wikipedia. (2024, May 14). Python. Wikipedia, La Enciclopedia Libre. https://es.wikipedia.org/wiki/Python*

29. *De La Paz, X. (2023, October 30). Primeros pasos con Python: ¡Hola, mundo! blog.vermiip.es. https://blog.vermiip.es/primeros-pasos-con-python-hola-mundo/*

30. *10 Librerías Python para Data Science y Machine Learning. (2023, March 17). Verne Academy. https://verneacademy.com/blog/articulos-ia/10-librerias-python-data-science-machine-learning/*

31. *Datacamp. (2024, May). Introducción a t-SNE. https://www.datacamp.com/es/tutorial/introduction-t-*

32. *Rodrigo, J. A. (n.d.). Selección de predictores, regularización ridge, lasso, elasticnet Y reducción de dimensionalidad. Ciencia de datos, teoría y ejemplos prácticos en R y Python. https://cienciadedatos.net/documentos/31_seleccion_de_predictores_subset_selection_ridge_lasso_dimension_reduction*

33. Adam | Interactive chaos. (n.d.). Interactive Chaos. https://interactivechaos.com/es/manual/tutorial-de-machine-learning/adam